TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

AKI MÄKINEN

VISUALIZATION OF MEDICAL DATA USING WEB TECHNOLOGIES

Master's Thesis

Examiner: Professor Tommi Mikkonen
Examiner and topic approved by the Faculty Council of Computing and Electrical Engineering 9th of December 2015

ABSTRACT

Aki Mäkinen: Visualization of medical data using web technologies
Tampere University of Technology
Master of Science Thesis, 53 pages
June 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen

Keywords: Web, patient monitoring, HTML5, canvas, JavaScript, AngularJS

Over the years, the medical expenses caused by inpatients have been increasing all around the world. One reason for this is increase in treatment costs. Some new medicines and methods may be cheaper, while some can cost significantly more. Unless hospital funding is increased accordingly, ways to save money must be found.

One way to reduce the costs and free hospital resources is to discharge patient as soon as possible. Problem in this is how to do it without risking the patient's health and possibly life by discharging her too early. After patients are discharged, they also are almost invisible to hospital systems, making it hard to monitor their recovery. Traditionally monitoring the recovery means patient visiting the hospital for checkup, or possibly counselling via phone. Checkups, however, can put strain on the patient especially if she lives further away from hospital, while through phone counselling it is hard to get full grasp on the situation. To solve these problems, patient remote monitoring solutions have been developed. Remote monitoring allows the patient to stay at home while a doctor or a nurse monitors her vital signs and other data at the hospital.

In this thesis, a patient remote monitoring application for demonstration purposes was implemented using modern web technologies. The application was written mostly in JavaScript, using AngularJS framework, cascading style sheets and HTML5's canvas element and server-sent events. Bootstrap CSS and JavaScript framework was also used to some extent. A generic Internet of Things cloud was used to store data and to retrieve it. Additionally, the data sent to the cloud is relayed to the application using server-sent events. For the evaluation, mocked sensor data was sent to the cloud back-end. The results were mostly positive. In extreme situation AngularJS begun to slow down and depending on the platform and setup, the CPU usage of the canvas rendering using the custom visualization library was considered too high. The end-to-end latency was mostly good, though because of occasional latency spikes, it cannot be used in critical situations where latency must be consistent. Overall, the application and the end-to-end system worked well.

The work can be considered successful. Even though the application has some performance issues, some of them are very unlikely to occur in real usage. Therefore, it can be said that the application performed well and achieved its goal. It provides a good basis for further development and optimizations and proves that web technologies can be used in medical domain, possibly even in hospital wards.

TIIVISTELMÄ

Aki Mäkinen: Lääketieteellisen tiedon visualisointi web-teknologioin
Tampereen teknillinen yliopisto
Diplomityö, 53 sivua
Kesäkuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: Web, potilasmonitorointi, HTML5, canvas, JavaScript, AngularJS

Sairaalahoidossa olevista potilaista aiheutuvat kustannukset ovat olleet nousussa jo vuosien ajan ympäri maailman. Eräs syy tähän on hoitomenetelmien kustannusten nousu. Osa uusista lääkkeistä voi olla halvempia verrattuna entisiin, mutta osa saattaa maksaa selvästi enemmän. Ellei sairaaloiden rahoitusta lisätä, on löydettävä keinoja pienentää menoja.

Eräs tapa pienentää kustannuksia ja vapauttaa sairaaloiden resursseja on kotiuttaa potilaat mahdollisimman aikaisessa vaiheessa. Samalla tulee pitää huolta, että potilaan terveyttä tai henkeä ei vaaranneta kotiuttamalla hänet liian aikaisin. Kotiuttamisen jälkeen potilaat ovat miltei näkymättömiä sairaalajärjestelmille, vaikeuttaen toipumisen seurantaa. Perinteisesti toipumista valvotaan kontrollikäyntien avulla, tai joissain tilanteissa puhelinneuvonnan avulla. Kontrollikäynnit voivat kuitenkin rasittaa potilasta, etenkin jos hän asuu kaukana sairaalasta, jolloin kulkeminen voi olla hankalaa. Puhelinneuvonnalla on puolestaan vaikea saada kokonaiskuvaa potilaan tilasta, vaikka se salliikin potilaan pysymisen kotona. Näiden ongelmien ratkaisemiseksi on kehitetty etämonitorointimenetelmiä, joiden avulla sairaalahenkilöstön on mahdollista tarkkailla potilaan elintoimintoja, vaikka potilas olisikin sairaalan ulkopuolella.

Tässä diplomityössä toteutettiin demonstraatiotarkoituksiin potilaan etämonitorointiin käytettävä sovellus käyttäen moderneja web-teknologioita. Sovellus toteutettiin pääsääntöisesti JavaScriptillä, käyttäen AngularJS -sovelluskehystä. Muita käytettyjä teknologioita olivat cascading style sheets, HTML5, canvas ja server-sent events. Myös Bootstrap CSS- ja JavaScript -sovelluskehystä käytettiin tiettyjen ominaisuuksien toteutukseen. Geneeristä esineiden internetiin tarkoitettua pilvipalvelua käytettiin datalähteenä ja – varastona, sekä syöttämään reaaliaikadataa monitorointisovellukselle. Tulokset olivat enimmäkseen positiivisia. Ääritilanteissa AngularJS alkoi hidastaa sovelluksen toimintaa. Tämän lisäksi, riippuen ajoympäristöstä, canvas-elementille piirtäminen nosti suorittimen käyttöasteen liian korkeaksi. Järjestelmän viive päästä päähän oli pääsääntöisesti hyvä, mutta ajoittaisten korkeiden latenssien vuoksi sitä ei voi käyttää tilanteissa joissa viipeen tulee olla mahdollisimman alhainen ja tasainen. Kaikkenaan sovelluksen ja järjestelmän katsottiin toimivan hyvin.

Suorituskykyongelmista huolimatta työ oli onnistunut. Havaitut ongelmat ovat hyvin epätodennäköisiä todellisissa käyttötilanteissa ja sovellus toimi yleisesti ottaen hyvin saavuttaen sille asetetut tavoitteet. Näin ollen sitä voidaan käyttää pohjana jatkokehityksessä. Työ osoittaa, että web-teknologioita voidaan käyttää lääketieteellisen tiedon visualisointiin, mahdollisesti jopa sairaaloiden osastoilla.

PREFACE

This thesis has taken me to uncharted waters. The medical domain was far from familiar to me and even now that this work is done, I have only scratched the surface. However, it has been a great joy to do this, as this kind of work is why I chose this field. It is now time to end one journey, and begin a new one.

I want to thank all who have supported me throughout the project. To my parents, thank you for being there for me the whole time. Thanks for my friends for giving me something else to think about every now and then. My colleagues, I want to thank you for supporting me while working and writing this thesis. Finally, a big thank you to Tommi Mikkonen for great guidance and patience.

Tampere 24.5.2016

Aki Mäkinen

TABLE OF CONTENTS

1. INTRODUCTION

For years, the medical expenses have been rising [1]. Every inpatient is an increasing expense to the hospital and something needs to be done. One solution is to discharge patients as soon as possible, which can be challenging, as it must be made certain that it is safe to do so. Outside the hospitals, the recovering patients have very limited visibility to the hospital systems making it difficult to monitor the recovery. While it is possible to make an appointment for a checkup, it may put strain on the patient especially if she lives far from the hospital. In addition, making the appointment so that it does not conflict with schedules otherwise can be challenging. One increasingly popular solution to this is remote monitoring. Remote monitoring makes it possible for example to monitor the patient in real-time, have the data upload happen automatically or have control on when the measurements are done if actuation is implemented.

The goal of this thesis is to prove that web technologies can be used to implement a patient remote monitoring system. The focus is on implementing the monitoring application, which done using JavaScript and hypertext markup language (HTML) version 5 (HTML5). The technologies chosen for the application are AngularJS framework, HTML5, canvas element for graphs, cascading style sheets for styles and server-sent events for receiving live data from the back-end. As a back-end, a generic Internet of Things (IoT) cloud service is used. The back-end receives the data from sensor, stores it and sends to any applications that has registered as a listener. The back-end also has capabilities for data analytics. Because the application's goal is to study the use and viability of the chosen technologies in patient monitoring, medical standards and evaluation of user experience and usability has been left outside the scope of this thesis. This is also to limit the scope of the thesis, as it would otherwise become too broad.

Chapter 2 introduces the basics of patient monitoring, what remote monitoring is, what current solutions there currently exists and what data formats are currently in use. Chapter 3 goes through the technologies that are relevant to this work. Chapter 4 looks into the architectures and design principles used in the system. Chapter 5 goes through the end-to-end system and the application, and Chapter 6 presents the evaluation of the technologies used in the system and ideas for future development. Finally, in Chapter 7 are the conclusions.

2. PROBLEM DOMAIN DESCRIPTION

Healthcare is evolving in many ways. New drugs and treatment methods are developed constantly while bacteria and viruses are adapting to existing medication. In addition to treatment, hospitals, interaction with patients and ways to diagnose illnesses are changing as well. In the field of computer science, this evolution means studying and improving user experience for both the doctor and the patient, designing new machinery and algorithms and creating software to aid the whole process. This development can mean shorter stays in hospitals, but in some case it may also increase the expenses [2]. While the technology advances and improves the efficiency along with other aspects of healthcare, the human as a user is always needed and is an important part of the whole system [3] 4].

Section 2.1 describes shortly partially through examples what traditional patient monitoring is. Section 2.2 discusses on what remote monitoring is, its history and its benefits to patients and to hospitals. Section 2.3 takes a quick, superficial look at few current companies that provide remote monitoring solutions. Section 2.4 introduces two current or upcoming data formats used with medical data.

2.1 Patient monitoring

The classical case of patient monitoring is the hospital ward, where the patient's state is monitored by attaching different sensors to her. The patient is mostly stationary and therefore it is not necessary for the monitoring equipment to have good mobility. This does make it possible to eliminate most of the latencies and other problems that might occur with different solutions.

Inside the hospital, the monitoring system has high requirements for its functionalities. The system must be as close to real time as possible, measure and visualize the data accurately and work reliably since inaccurate data, too long delays and malfunctions may lead to death in hospital environment. The devices in the hospital measure at least the primary vital signs, which are blood pressure (BP), respiration rate (RR), body temperature (BT) and heart rate (HR) [5].

Figure 1 presents two examples on what patient monitoring system displays in hospitals. Both devices are capable of showing measurements and derived values such as ECG, airway gases and pulse oximetry. It is unknown whether respiration rate is visible generally, as only Philips lists it under “Neonatal Monitoring Features” in the IntelliVue MX800 Technical Data Sheet. The IntelliVue MX 800 can display 12 waves simultaneously, while B40 Patient Monitor can display only six. Thus it is possible to display all ECG waves in 12-lead ECG monitoring. [6][7]

Figure 1. Philips IntelliVue MX800 (left) and GE Healthcare B40 Patient Monitor (right) [6][8]

Patient bedside monitoring systems are also capable of raising an alarm when a defined limit is exceeded. This is a critical feature in order for the hospital staff to be able to respond to critical situations quickly. Both devices are capable of generating alarms and allow the limits to be customized [6] [8]. The alarm can be visual and/or audial and may be local as well as remote, sending the alarm to a monitoring center [9].

2.2 Remote patient monitoring

Although comfortable and familiar environment can have a positive effect on healing process, it is not the only reason why there is so much interest in remote monitoring [10]. Hospitals are aiming to improve resource allocation efficiency and to reduce personnel workloads, waiting times and costs both for the patient and to the hospital.

Medical care costs have slowly been rising globally as visible in Figure 2. This puts pressure on hospitals to discharge patients as soon as safely possible and to keep them from readmitting using follow-ups and other means to follow the patient's situation. [1]

With remote monitoring, doctors and nurses can check the patient's status when she is out of the hospital, which means the patient does not necessarily have to go to hospital for checkup. This benefits the hospital, as resources can be reallocated, the patient and the society when taking sick leaves into account [11]. By using remote monitoring, hospitals can make sure the patient is following the given regimen and if needed, alter it or remind of it. This helps to avoid readmission, which e.g. in U.S.A. results in penalties imposed by the Medicare if the certain readmission ratio is exceeded [12].

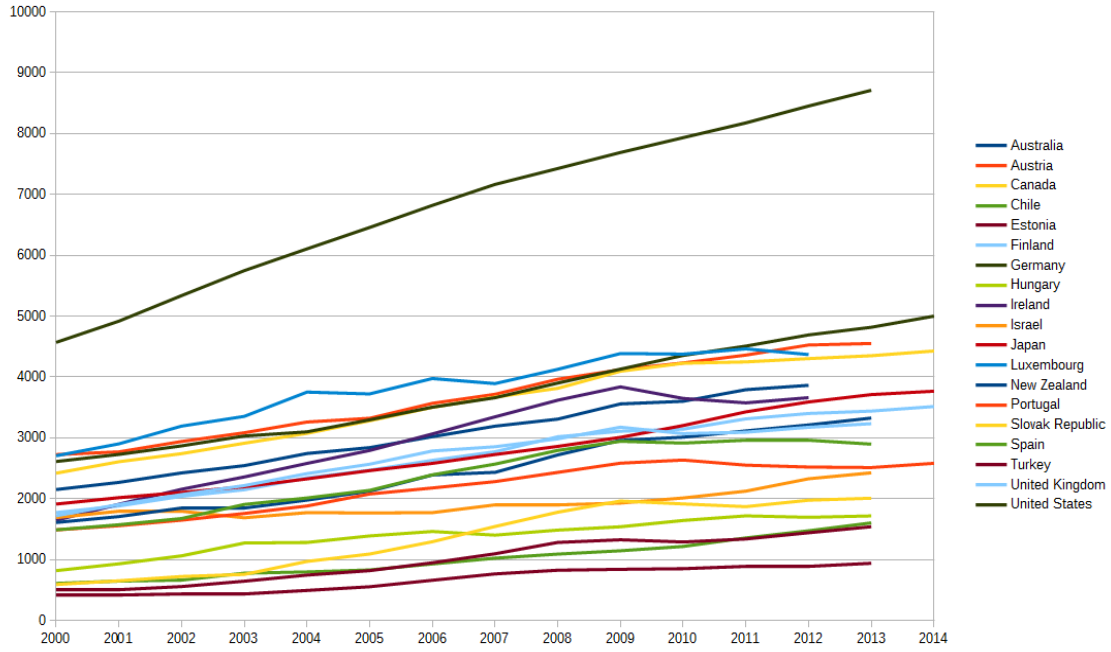

Figure 2. Current expenditure on inpatient health care per capita. Current prices and current PPPs. [1]

Patient remote monitoring can be traced to the 1970's, when transtelephonic monitoring (TTM) was first designed and implemented to monitor pacemakers. During the next decades, TTM was expanded and it became common in the United States of America, but not in Europe [13]. In TTM, the connection is formed via phone line, allowing device diagnostics and sending small amount of sensor data. Compared to remote monitoring solutions, which can store historical data and send real time data, TTM offers very limited way to monitor patient remotely. In addition, most TTM solutions require patient interaction to store samples and to send them, while remote monitoring devices can be set up to do both of these automatically. If the interaction requires directions from a doctor or a nurse, sending data via TTM can be especially problematic for those with impaired vision or hearing. [14]

Some risks of the remote monitoring are networking issues, battery life, power consumption, and sensors staying put. In all cases, networking can cause issues, even though the coverage of cellular networks such as 2G, 3G and LTE (4G) is good in most locations, but there are still areas where it might not be possible to form a connection to the network. Even if the area has good coverage and good network quality most of the time, all networks are still susceptible for interference and faults, wireless networks more than cable connections. In addition, bad network quality may result in slow connection and inability to monitor the patient in real time. Battery life is also important, as in most cases the patient must be able to function as normally as possible which limits how often the device can be connected to the charger and of course how long it takes to charge. As for the safety of the patient, the remote monitoring has been proven a safe method of monitoring

recovering heart patients in the Lumos-T Safely Reduces Routine Office Device Follow-Up (TRUST) trial [15].

2.3 Current solutions

Today several companies offer remote patient monitoring services and products. Most of these do not, however, have real time monitoring or emphasize it in their product. For example, Silvermedia (Poland), Comarch (Poland), Medixine (Finland) and Medtronic (U.S.A.) are developing and providing remote monitoring devices currently. Out of the four companies, only Comarch clearly states that real time monitoring is supported. Common to all four companies is that they support several types of data, but put more focus on ECG and cardiovascular health. Silvermedia and Comarch give the data for specialists to analyze. Comarch also emphasizes the development of analysis algorithms such as those used in cardiology, audiology, allergology and neurology. It is also noticeable that at least Medixine and Silvermedia offer counselling through the service by using the application in the patient's smartphone. [16] [17] [18] [19]

As an example, Table 1 shows Medtronic's specification on displayed data in their ZephyrLife Home Portal Interface. Most of the data, such as heart rate, respiratory rate, blood pressure and ECG can be considered standard data shown in most remote monitoring systems. In addition, the system displays patient's weight and glucose, both gathered from devices connected with Bluetooth. It should be noticed that the system also supports alerts and configurable boundaries for them. It is unknown if the other systems have alert functionality. [19]

Figure 3 shows the high-level architecture of Medtronic's remote monitoring system and similar structure is in use in many generic Internet of Things solutions. Patient has one or more sensors attached and transmitting data to a specialized device or a smartphone, which in turn transmits the data to the service provider's server, either in batcher or, as in some cases, real time. On server, it is possible to analyze the data and recognize some medical conditions. From the server the data is sent again to a client software either when it is requested or immediately if real time monitoring is available. [21]

Table 1. Data displayed in ZephyrLife Home Portal Interface. [20]

Patient details:	Name, Address, Phone, all sensor IDs, caregiver, emergency contact
Heart Rate:	Beats per minute
Respiratory Rate:	Breaths per minute
Posture/Activity:	Icons indicate lying, upright, walking and running
Temperature:	Bluetooth Thermometer °F
Last Update:	hh:mm of last data update
Blood Pressure:	Systolic/diastolic; Bluetooth blood pressure
Saturated Blood Oxygen:	%; Bluetooth SpO_2
Weight:	lbs; Bluetooth Weight Scale
Glucose:	mg/dL; Bluetooth Glucometer
BioModule Battery Level:	% of remaining charge
HealthHub Battery Level:	% of remaining charge
ECG:	30 second capture on demand or on alert, thumbnail gallery and event marker
Historic Data:	All of the above parameters in tabular format
Trend Data:	All of the above parameters in trend form
Alert Levels:	Upper & lower configurable levels, posture transitions and turn time
Alert Notifications:	Color coded indication on display and audible alerts
Security:	Secure Bluetooth connection, secure portal log in

Figure 3. Medtronic remote monitoring system end-to-end architecture. [21]

2.4 Current data formats for medical data transfer

The field of data format for medical data transfer and storage is very heterogeneous at the moment. Naturally, some of the formats are competing standards while others are designed for specific purposes. For example, binary formats are good for low-level communication between devices, or when the information must be concise. Formats based on extensible markup language (XML) or JavaScript Object Notation (JSON) are both human- and machine-readable and easier to handle in software. JSON-based formats are still rare in medical use but this is slowly changing as formats like HFIR are introduced to the field [22].

In [23], Trigo *et al.* go over 38 data formats that either are designed for ECG data, or support it otherwise. Even though certain data formats such as EDF(+) have become widely used and supported, many hardware manufacturers still define and use their own formats at least internally for storage and transfer [23]. One reason for this may be the fact that commonly used formats are for either storage or importing and exporting larger amounts of data. Because of this, they may contain properties that are not necessary in data transfer from the measurement devices to a monitor or a server. In internal use, not everything in patient's personal data is needed and even just the sensor data can be enough in certain situations if no deep analysis based on different patient parameters is required. Thus, the data format can be optimized to fit the needs of the application. For example, for transfer more concise format is better especially when sending data from the remote monitoring device to a server.

European Data Format. The first version of European Data Format (EDF) was published in 1992. The need for it arose when the participants in the project "Methodology for the Analysis of the Sleep-Wakefulness Continuum" needed to share sleep recordings with each other [24]. The first version was very simple and did not contain support for annotations or events. Support for them, stimuli and interrupted recordings was added in

EDF+, which was published in 2003. The additions to the format made it possible to store any medical recordings while making it stricter than the first version. EDF(+) was designed for data storage but because it is relatively concise format and considering the speeds of internet connections today, it could be used to transfer data continuously. The data file consists of a header record and a data record. In the header record is stored the necessary information to identify the patient and some technical information. The measurement data and the metadata related to it is stored in the data record. These days EDF is one of the most commonly used data formats though it is not standardized. [25]

Fast Health Interoperability Resources. Fast Health Interoperability Resources (FHIR) is a standard in development by the Health Level Seven International, a not-for-profit organization. Health Level Seven International has also developed and published HL7 version 2 and HL7 version 3 standards, latter in use in the Finnish National Archive for Health Information, Kanta. [26] [27]

FHIR consists of resources. Each resource has generic metadata, a human readable portion, a URL, a set of common elements and a representation. Generic metadata contains information about the resource. Accepted fields are `versionId`, `lastUpdated`, `profile`, `security` and `tag`, each of them being optional [28]. The human readable portion can contain extensible hypertext markup language (XHTML) with some restrictions, such as `head` and `body`, out of security reasons [29]. The representation is the format used in the document. The specification describes two formats, JSON and XML, but does not limit the data format to them, which means the format is well suited for REST services. [30]

A resource has always a type. Currently the main categories are clinical, identification, workflow, infrastructure, conformance and financial. Each of these contain four subcategories that have from one to seven resource definitions. [31] The standard is in itself extensive, covering several fields, and if needed, it can be further expanded by using extension element or modifier extension.

FHIR is currently under development. The latest version is Draft Standard for Trial Use 2 (DSTU2). This means that it does not have to follow HL7's Inter-version Compatibility Rules until it reaches *Normative* status. [32]

3. INTRODUCTION TO WEB TECHNOLOGIES

Ever since the invention of computers and the first general purpose computers, the field of computation has evolved rapidly and in the 1960's it led to the development of Advanced Research Projects Agency Network (ARPANET), the predecessor of modern Internet. In the 1970's the next step was taken, when Transmission Control Protocol / Internet Protocol (TCP/IP) was designed and published. The continuous development finally led to the introduction of Domain Name System (DNS) in the 1980's. Everything was ready for the World Wide Web (WWW). [33]

Although the Internet and especially the World Wide Web are relatively young concepts and inventions even on the field of computing, the role of web technologies in everyday life has become very important. This chapter contains a brief introduction to web technologies relevant to the thesis' subject.

Section 3.1 covers the main idea behind the protocol used by the World Wide Web. Section 3.2 introduces the latest major version of HTML and looks at two important features of it. Sections 3.3 and 3.4 take a closer look to two HTML5's technologies: canvas and server-sent events. Section 3.5 goes through the basic principles of Cascading Style Sheets. Section 3.6 covers the basics of JavaScript. In Section 3.7 is a short history of JavaScript Object Notation and the format explained. Section 3.8 explains what is responsive web design and Bootstrap.

3.1 HTTP

While the Internet Protocol Suite takes care of the data transfer, an application layer protocol is used on top of them to specify the interface and the protocol for the communication. HyperText Transfer Protocol (HTTP) was designed to be a simple application layer that could be used to transfer HTML files. [34]

The first version of the HTTP was 0.9 in 1991. This version was still very simple and could only be used to transfer HTML files. However, it did define the basic principles of HTTP, which are still used. HTTP/0.9 specified how the connection is made, what is the request and response, and what happens after the transfer is complete. Connection was to be formed by using either domain name or IP address and a port number, which defaulted to 80 if not defined explicitly. It was also stated that the transport layer protocol did not need to be TCP, but others could also be used. [34]

In HTTP/0.9, the request contains a single line of ASCII characters consisting of word "GET" and the address of the document, separated by a single space. The line should be terminated by `CR LF (Carriage Return, Line Feed)`. With this, the protocol defined the

first HTTP method, which was to be joined by several others in later versions. In response to the request, a HTML file is returned as a byte stream containing ASCII characters. Each line in the response can vary in length, but a length of 80 characters is recommended. After the response is transferred, the connection is terminated and the server is not required to store information related to the request. This applies even if the connection is terminated early by the client, thus making the HTTP a stateless protocol. [34]

Soon after HTTP/0.9 was defined, the development of HTTP/1.0 began and it was released in 1996 in Request for Comments (RFC) 1945. In addition to "GET", it defined several new methods: PUT, POST, DELETE, HEAD, LINK and UNLINK. With this new version, new response status codes were also introduced, bringing the standard closer to the current state but still missing several status codes as well as some less known methods. In this project, the most important methods are so called CRUD (create, request, update, delete) methods: POST, GET, PUT and DELETE. [35] In 1999, HTTP version 1.1 was released and it added many new features to the protocol. A comprehensive list of HTTP methods is maintained by Internet Assigned Numbers Authority (IANA) at [36]. Currently HTTP/2 is in development. [37] [38]

Message structure. The basic format and structure of a HTTP request and response that is still in use was defined already in the first draft of the protocol. Originally, the header was the only part in requests, and responses contained only the HTML document. It was quickly noticed that additional information had to be relayed in the header section and that it was necessary to add a header in the response as well. Requests contain at least a request line that has the method, resource path and HTTP version. In HTTP/1.1, a `Host` header line with the host string is also required. Additional header lines are used to transfer information such as Multipurpose Internet Mail Extensions (MIME) type or authentication data to the server. Responses contain at least a status line that contains the used HTTP version and a status code, other headers being optional. [39]

Authentication. An important feature required by many websites is authentication. HTTP provides Basic Access Authentication and Digest methods for this purpose. In Basic authentication, username and password are concatenated to a single string where they are separated by colon, encoded with Base64 and added to `Authentication` header with the information of what method is used. An example of `Authentication` header when username is "*tteekkari*" and password is "*verysecret*": `Authentication: Basic dHRlZWtrYXJpOnZlcnlzZWNyZXQ=`. [40] HTTP Basic authentication alone is very insecure method as all the content is unencrypted and Base64 encoding is trivial to decode. This security problem can be solved by using HTTP over SSL (HTTPS), which encrypts all the communication between the client and the server.

HTTP Digest authentication is based on challenge-response model. When a user requests a resource without credentials, the server response is a challenge with 401 HTTP status code to which the client must answer to gain access to the resource. The challenge has a

WWW-Authenticate header, which contains the necessary information for the client to answer to the challenge. Table 2 contains what parameters the WWW-Authenticate header may contain and their descriptions. These parameters are used on the client either as-is in response to the challenge, to produce another value or as information to the user. When responding to the server, the client sends a request with Authorization header. [41]

Table 2. The WWW-Authenticate response header field parameters. [41]

Parameter	**Description**
Realm	String that usually contains at least host name of the authentication service or other information from which the user can tell what username-password combination to use.
Domain	List of URIs that describes what resources the user can access.
Nonce	Server generated unique string that is used only once. Base64 encoded or hexadecimal data
Opaque	String generated by the server which the client should send back to the server in its response to the challenge
Stale	Flag that informs whether the username and/or password the user gave were correct or if new ones should be re-prompted.
Algorithm	String that contains the hashing algorithm to use. Valid values are SHA-256, SHA-512-256 and MD5. It is recommended to use SHA-256 algorithm, though MD5 is the default if no algorithm is defined.
Qop	String consisting at least of one token that tells what quality of protection values the server supports. This parameter is always required.
Charset	Optional parameter to inform the client what character encoding the server supports.
userhash	Informs the client whether the server supports username hashing or not. If this parameter is not defined, then it is supposed to be false.

Table 3 contains the parameters allowed in the Authorization header. The response is produced by hashing string Hash1 ”:” nonce ”:” nc ”:” cnonce ”:” qop ”:” Hash2 with the chosen algorithm. In this string Hash1 is produced by hashing string username ”:” realm ”:” passwd and Hash2 by hashing string Method ”:” request-uri if qop value is ”auth” or Method ”:” request's URI ”:” Hash of entity-body, if qop is ”auth-int”. [40]

Because username and password are hashed in HTTP digest authentication, it is more secure than the basic authentication. With the use of nonce values, it also prevents replay attacks, where the data is retransmitted with malicious intentions. [41]

Table 3. The authorization header field parameters. [41]

Parameter	**Description**
`Response`	A string consisting of hexadecimal digits.
`Username or username*`	Parameter used to send the username to the server. If username hashing is not used and username contains characters outside of the specified set, then the latter form is used.
`Realm`	Same as in WWW-Authenticate header.
`Uri`	Requested URI.
`Qop`	Quality of protection applied to the message.
`Cnonce`	Cryptographic nonce generated by the client. Can contain only ASCII characters. Always required.
`Nc`	Count of requests sent by the client using the current nonce. In hexadecimal format.
`userhash`	Parameter that tells whether username hashing is used. Can be either ”true” or ”false”

The third well-known form of authentication is token-based authentication, where username and password are exchanged to an authentication token, which is valid for limited time. With single sign-on (SSO) services, token-based authentication has become more popular option. Most commonly used way to authenticate the user with token works very similarly to Basic authentication. Header's content consists of word "Bearer" followed by the given authentication token, e.g. `Authentication: Bearer 8e9f5d38-94f0-42a0-a834-c9aedb172af6`. The token is issued by an authentication server to which the username and password are sent. This is done only if the client does not have a token or the token has expired. Because the token works as a replacement for username and password and is sent in plaintext, the same security concern with Basic authentication applies as well as the solution to it. The benefit with token-based authentication is that with external authentication, the same credentials can be used in several services. As the tokens can be revoked and can expire, the possible damage can be mitigated if a valid token ends up to a malignant operator. As with basic authentication scheme, HTTPS should be used in token-based authentication. [42]

3.2 HTML5

HTML (HyperText Markup Language) is the markup language designed to describe the content of a web page. The first version was originally designed by Tim Berners-Lee and it was first used by the WorldWideWeb, the world's first web browser. The first version of HTML was based on Standard Generalized Mark-up Language (SGML) and since HTML 2.0, it has been defined with SGML. [43] HTML5 is the fifth major version of the language and it became a World Wide Web Consortium's (W3C) recommendation in late 2014 [44]. While HTML itself is a markup language, the term HTML5 is often used to refer a collection of modern web technologies.

HTML defines a document. The document usually starts with a document type declaration and while it is not required, it is strongly recommended as it helps the browser to ignore the unsupported features. In HTML5 this is done by entering `<!DOCTYPE html>` on top of the document. This was much more verbose on previous versions of HTML and was shortened to HTML5 [45]. After document type declaration, the actual content is inserted inside `<html>` element. The content of the html-element can be split to two parts, header section and a body. Header section is defined using `<head>` element, contains metadata about the document, and allows linking scripts and stylesheets to the document. Metadata can be used to declare the character set of the document with a `meta` element: `<meta charset="ISO-8859-15">`. As with the document type, the character set declaration was simplified from the previous versions. [44]

HTML5 introduced several new elements as well. Canvas element for 2D and 3D graphics was added, as well as inline Scalable Vector Graphics (SVG) support. In addition, new semantic and structural elements have been included with new form elements, input types, input attributes and media elements. [44]

An important part of web programming is the concept of Document Object Model (DOM). As stated at W3C's website, *"The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. [46]"* DOM also offers a way to organize the nodes of documents, resulting in a tree structure where each node may have a parent node and several child nodes. [46] In web programming, these nodes are most commonly accessed using JavaScript. Modifying the DOM is not, however, a light task and having multiple DOM updates occurring often can slow down the application easily. Today on desktop computers, this does not become a problem easily, but it still can be one on mobile devices.

The current HTML is not developed only by W3C. The Web Hypertext Application Technology Working Group (WHATWG) founded by people from Apple, the Mozilla foundation and Opera Software is developing actively HTML Living Standard. The difference between W3C's and WHATWG's work is that WHATWG's HTML Living Standard is

actively updating and it does not define different versions, while W3C aims at more static, complete standard with versioning. HTML5 contains work done by both groups and W3C takes fixes made by WHATWG to the HTML5 standard. [47]

3.3 Canvas element

The 2D Context for canvas element became a W3C recommendation 19th of November 2015. It provides an interface to a canvas element for drawing and manipulating graphics. The functionality is not limited only to drawing and showing two-dimensional images on the canvas, but it can also be used for showing and manipulating videos. [48]

Canvas offers a low-level solution for drawing shapes and images dynamically on a HTML document. It uses raster graphics for drawing, which makes image scaling more difficult than when using vectors. The information on how the content of the element is drawn must be stored in user defined data structures as the element does not store what was drawn and how. This is different from SVG, which uses DOM to store the shapes as objects delegating the actual drawing to the browser. It also makes it easier to manipulate already existing elements. However, when plotting a large amount of data, this also makes it somewhat slower and heavier especially if the data cannot be drawn as a single vector, which is an important thing to notice when working with real time data. If the data is mostly static and user interaction is required (e.g. hyperlinks), SVG is easier to use. On the other hand, canvas has an advantage over SVG when it comes to custom animations and control over the rendering procedure. [49]

3.4 Server-Sent Events

Real time communication between the server and the client has traditionally been problematic because of the stateless nature of HTTP. This limitation has been circumvented with methods such as long polling, where the server will not respond until data is available and a new request is made after one is resolved. In 2011, WebSocket technology was standardized and released as RFC 6455. WebSockets allow two-way communication between server and client over TCP connection. [50]

HTML5 introduced a new mechanism for real time communication between the server and the client, Server-Sent Events (SSE). The idea behind SSEs is, that client registers to listen SSE events and server then pushes data to the client when needed. [51] The formed connection allows traffic only to one direction, so communication from client to the server has to be implemented otherwise. This resembles long polling, but the difference is that SSE connection is persistent and a new connection is not required after data is received.

A new SSE connection is formed using `EventSource()` constructor, which takes an URL as the first argument and an initialization dictionary as an optional second argument. If

the base URL can be resolved, a new `EventSource` object will be created, otherwise `SyntaxError` exception is thrown. By default, the Cross Origin Resource Sharing (CORS) is set to `anonymous`, but with the initialization dictionary containing attribute-value pair `withCredentials: true` it can be set to `Use Credentials`. This is necessary for example when using cookies, as otherwise they will not be sent in cross-site requests. When forming the connection, the server sends a HTTP response to the client with the necessary headers set, after this only the actual data is sent. This way the overhead created by the HTTP can be eliminated. The data consists of rows, each row containing one event as in the example above. [52]

Because SSE connection is one-directional, it is well suited for sending notifications to an application or for monitoring, where most of the data flows from the server to the client. With the ability of having several types of events in a single stream, the same connection can be used to deliver many types of data.

3.5 Cascading Style Sheets

The development of Cascading Style Sheets (CSS) began in 1994 from the need to add styles to HTML documents. CSS was not the first attempt to define a language for styling HTML, e.g. Robert Raisch had defined a style language already in 1993. CSS's advantage was that it took the display device and the browser into account in addition to reader's and author's design. CSS level 1 became a W3C recommendation at the end of 1996 and work on level 2 in W3C working group begun in 1997. CSS level 2 was released as a Recommendation in 1998. Both CSS level 1 and 2 were monolithic specifications but from level 3 forward, the specification has been modular. Each different module may contain new features and extensions to level 2 specification, but must maintain backward compatibility. Currently there are level 3 and level 4 modules under work in W3C. [53] [54]

The language syntax is very simple consisting of possible at-rules (e.g. `@document`, `@media`, `@page`), selectors and style definitions [55]. With `@media` it is possible to customize the page for different screen sizes or for printing and it is used in responsive web design.

Selectors at simplest are elements, element identifiers and element classes. Any element can be matched with the universal selector *. CSS defines some relationships for the selectors to give more control over the styles. Table 4 contains some most commonly used relationship based selectors. Out of these four, the third one is actually a pseudo-class added to a normal basic selector while the other three are combinators. In the table, only ~ combinator is not included.

Pseudo-classes can be added to the basic selectors. With them, it is possible to apply a style to an element when it is in certain state. For example, `:hover` pseudo-class applies

the rules to the element when cursor is on top of it, while with :focus, the rules are applied when the element has the focus.

Table 4. Mozilla Developer Network – Common selectors based on relationships [56]

Selector	Selects
`A E`	Any E element that is a *descendant* of an A element (that is: a child, or a child of a child,*etc*.)
`A > E`	Any E element that is a *child* (i.e. direct descendant) of an A element
`E:first-child`	Any E element that is the *first child* of its parent
`B + E`	Any E element that is the next *sibling* of a B element (that is: the next child of the same parent)

Rules are defined inside blocks that are marked with curly braces, the top-level selector or type definition being an exception. Table 5 lists the basic selectors used to reference to an element of certain type, elements with certain class or an element with specific id.

Table 5. Basic CSS selectors

Selector	Explanation
`#a`	Selector beginning with a hash matches an element with id "a"
`.b`	Selector beginning with a period matches all elements that have class "b"
`c`	Selector matches all elements of type "c"

In the Example 1, the first block defines rules that are applied when the viewing media is a screen with maximum width of 800 pixels. The rule that sets 200 pixels as the minimum width and 500 pixels as the maximum width of the element is applied to an element that has id `maincontent`. `@page` is an example of an at-rule and defines the margins when printing the document, in this case top- and bottom-margins are 0.5 inches and the side-margins are both 1 inch. The last rule is applied to all elements that have class `menuitem`, only when the cursor hovers on top of the element. [56]

Example 1. CSS rules and selectors

```
@media screen and (max-width: 800px) {
      #maincontent {
            min-width: 200px;
            max-width: 500px;
      }
}
@page {
      margin: 0.5in 1in 0.5in 1in;
}

.menuitem:hover {
      color: red;
}
```

Because the CSS allows presenting the document in different ways on different media, it can be used in web design to implement the web application only once, but still have it work well on various devices and screen sizes. This approach – responsive web design – is covered in Section 3.8.

3.6 JavaScript

JavaScript is a scripting language well known by developers these days. The first form of JavaScript, Mocha, was a prototype developed for Netscape web browser by Brendan Eich. It was soon renamed LiveScript and again JavaScript after a trademark license was received. [57]

Standardization of JavaScript began in 1996 at ECMA, which resulted in the release of the first edition of ECMA-262. At this point, the standardized version of JavaScript became known as ECMAScript and JavaScript as its implementation. Later on, second (1998) and third (1999) version of ECMAScript was released, third becoming the base for current JavaScript and JSON. Other better-known implementations of ECMAScript have been JScript.net and ActionScript 3 (AS3). The development of the fourth version was all but uniform and the first proposal was discarded. Second attempt was made to standardize AS3 features to ECMAScript, but because of disagreements and differences between it and JavaScript, the second attempt failed as well. After this, ECMAScript 3.1 was specified and released, and in 2009 renamed to ECMAScript 5. Sixth version was released in June 2015. [57] [58] [59]

As a language, JavaScript is much like many scripting languages, imperative with possibility for functional programming and using dynamic and duck typing. JavaScript can be thought to be object-oriented language, though until the latest version of ECMAScript, it has lacked the construct of a class. Object-oriented programming has been possible by using JavaScript objects, function variable scope or prototypes. Using functions for object-oriented programming is possible because JavaScript has two variable scopes: global

and function. Simulating classes and inheritance is possible in ECMAScript 5 in several ways. In functions it is possible to define properties and methods using `this` keyword or by using variables and return an object containing public variables and method. These two ways behave similarly outside the definition.

Inheritance can be implemented by setting properties and methods to the function's prototype. The methods set this way work as in the two earlier ways to define classes. Example 2 demonstrates how inheritance can be done in Javascript using prototypes and `Object.create` method. After defining `Example` and calling `ParentClass`' constructor inside it, `Example`'s prototype is set to `ParentClass`' prototype. In ECMAScript 6 classes can be defined using `class` keywords and extended with `extend`.

Example 2. Object oriented programming and inheritance in JavaScript

```
function Example() {
    ParentClass.call(this); // Call parent's constructor
    this.c = 100;
}

Example.prototype = Object.create(ParentClass.prototype);
Example.prototype.constructor = Example;

Example.prototype.actualPrint = function() {
    console.log(this.b / this.c);
}

var inheritance = new Example();
inheritance.setA(10);
inheritance.testPrint(); //Prints 31.41
inheritance.actualPrint(); // Prints 0.03141
```

JavaScript is these days used in many contexts, not only inside a web browser. It can be used as a scripting language in programs (e.g. Weechat [60]) and with Node.js [61] on server-side. On client-side, one of the most important features of JavaScript is the ability to manipulate DOM. This way the nodes can be changed, added or deleted. When combined with `XMLHttpRequest` (XHR), an API for sending HTTP requests by using client-side scripting languages like JavaScript, only parts of the web page can be updated and changed when needed. This technique is known as asynchronous JavaScript and XML (AJAX) when the data format used in communication between the server and the client is Extensible Markup Language (XML), or asynchronous JavaScript and JSON (AJAJ) if JavaScript Object Notation is used instead of XML. However, the latter term has not become as well accepted and widely spread as the first one, and the term AJAX is used in both cases instead.

3.7 JavaScript Object Notation

JavaScript Object Notation (JSON) is a data format based on ECMA-262 3rd Edition. It is simple format consisting of attribute-value pairs and thus is human-readable and easy to parse and generate programmatically in most programming languages. [62] Because of these properties, it is good for data interchange and has gained popularity at the expense of other formats like Extensible Markup Language (XML), which has been one of the most common formats and used for example in web programming in Asynchronous JavaScript and XML. Although JSON is less verbose and easier to parse and generate than XML, it lacks official support for schemas. As an attempt to fix this shortcoming, JSON Schema has been specified as informational Internet Engineering Task Force (IETF) Internet Draft, currently at fourth version [63]. JSON itself has been specified in two documents: ECMA-404 and RFC7159. The former of these is an ECMA standard and the latter is in IETF Standards Track as Proposed Standard. [64] [65] To take a closer look into JSON, the ECMA specification is used, as it is the original.

Format. JSON documents use unicode character encoding. At top level JSON contain always either an object delimited by curly brackets, or an array, which is delimited by square brackets. An object always contains attribute-value pairs where attribute is separated from the value with colon, while array on the other hand contains a list of values. In both cases comma works as a separator between the pairs or values. [62]

In addition to object and array, JSON has four other data types: `string`, `number`, `Boolean` and `null`. Boolean can be either `true` or `false` and is written without double quotes in JSON objects, the same way `null` is. As shown in Figure 4, a string can be empty or contain one or more characters. Quotation mark and reverse solidus has to be escaped inside a string. This is because the former is used as the string delimiter and the latter as it used as a control character inside a string. [62]

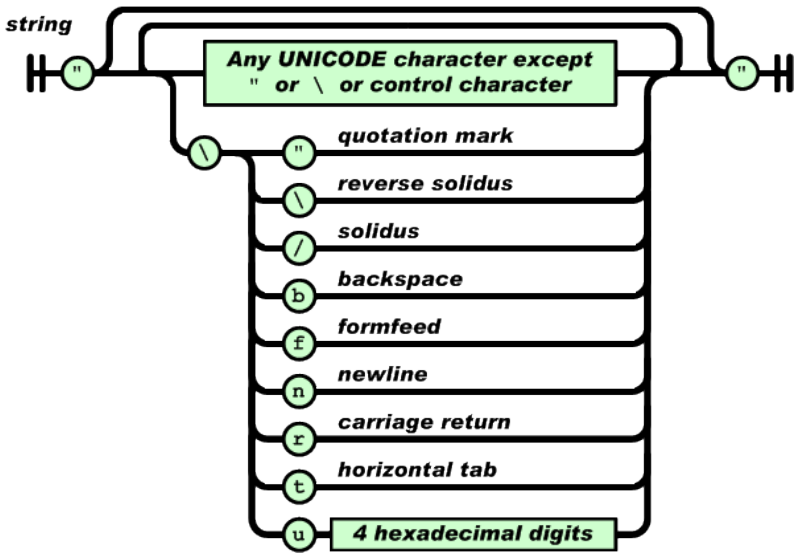

Figure 4. JSON string datatype definition [62]

Numbers are not delimited with any special characters as strings are. They may contain characters visible in the Figure 5 included *e* or *E* used in scientific notation, dot, plus and minus. [62]

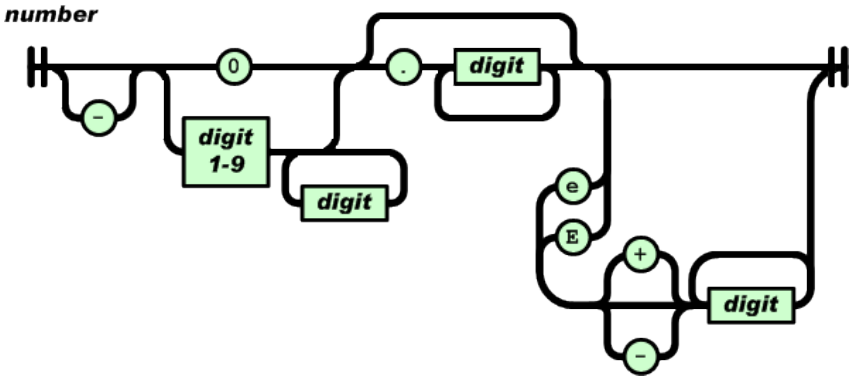

Figure 5. JSON number datatype definition [62]

3.8 Responsive web design

Mobile devices like smartphones and tablet computers have become commonly used in a daily life. While making the internet easily accessible from nearly everywhere, they do introduce problems when designing the content. Because of their size and different ways to interact with the user interface (UI), using the same design with them and a conventional computer deteriorates the user experience. One solution is to create separate UIs for each device but this increases the amount of work and makes maintaining the site harder. The goal of responsive web design is to eliminate the need for several versions of the site by making the site adapt to the device. The basic principles of responsive web design were introduced by Ethan Marcotte in [66]. According to him, the basic elements in responsive web design are fluid grids, flexible images and media queries. Fluid grids and flexible images adapt to changes in content and screen size and media queries are used for example to set styles according to the screen size. One way to bring responsive design easily to a web application is to use Bootstrap.

Bootstrap. Bootstrap is an open source CSS and JavaScript framework for creating responsive web sites. It was originally created for Twitter and before going open source, it was known as Twitter Bootstrap. The framework has been rewritten twice, and a third rewrite, Bootstrap 4 is under development. The latest stable major version of the framework is 3. The main features are implemented in CSS but some functionalities require the use of JavaScript. Bootstrap's CSS source code is available in both Less and Sass, which are CSS preprocessors designed to give better control over the CSS. The source codes must be compiled to pure CSS before they can be used on a web site. Bootstrap provides tools for several features used in responsive web design, such as grid layout that adapts to the screen size and responsive UI components. [67]

4. ARCHITECTURAL PATTERNS AND FRAMEWORKS

There are established patterns that are used in web development. Some of these patterns describe how applications communicate with the server or with each other's, while others describe the application architecture. In this chapter, the most relevant architectural patterns and styles are introduced in addition to AngularJS web framework.

The Section 4.1 delves into the basic architectural model behind the Internet and many other technologies, the client-server model. Section 4.2 presents representational state transfer, a popular architectural style used in web applications. Three commonly used architectural patterns used in many web frameworks and their relevance to this work is conversed in Section 4.3. Finally, in Section 4.4 one very popular web framework, AngularJS is presented.

4.1 Client – Server model

Client-Server model is the basic architectural structure for distributed applications. In addition to the client, another software is run either locally on the same machine as the client, on in another location on different machine. In the client-server model, client's operation is dependent on the server's resources and services. [68]

The method of communication can be almost anything, as the architecture model does not limit it. When running client and server on the same machine, the communication can happen for example by using file socket, but if the software is run on different locations, it must happen by some other means. Though the data can be transferred e.g. universal serial bus (USB) cable, the most visible way today for most people is the Internet using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The communication follows request-response pattern: for each request made by the client, the server sends a response.

When the communication happens over the Internet, the client-server architecture makes resource sharing easier and it improves accessibility, as the server can be made reachable from anywhere. By implementing and publishing an application programming interface (API) client can call it and the operation is run on the server instead of the client. Server can also implement roles for different users and limit access to the resources and the API, thus making it more secure. The centralized nature improves maintainability and the API works as an abstraction layer that hides the server side implementation from the client. This way changes to the server application can be made without changing the client. The

downside of the model is a reduction in robustness, at least when communicating between different machines. [68]

HyperText Transfer Protocol used in the data transfer in World Wide Web is designed on top of the client-server architecture. It is thus an important part of the WWW and modern web applications.

4.2 Representational State Transfer

Representational State Transfer (REST) is an architectural style commonly used by web applications these days. It was introduced by Roy Fielding in his Ph.D. dissertation "Architectural Styles and the Design of Network-based Software Architectures" and can be thought to be an alternative to Simple Object Access Protocol (SOAP). [69]

REST is a stateless client-server architecture, meaning that the communication happens always between a client and a server and it must be stateless. Because of this, each request done by the client must contain sufficient amount of information so that the server is able to perform the operation and respond to the request. Any information related to the state of the session must be stored on the client. Other constraints set by Fielding in his dissertation are cache, uniform interface, layered system and code-on-demand. [69]

Cache. The use of cache increases the efficiency of the network but at the same time requires that the data contained by the response can be marked as cacheable or non-cacheable. While this improves efficiency and scalability of the system, it may lead into situations where the data in the cache and the data in the response with a new request can differ. [69]

Uniform interface. Although genericity is a key point when designing software components, it is especially emphasized in REST architecture. Uniform interface's goal is to keep provided services apart from applications. Application specific interface may improve performance of a single application or in some cases a group of applications, but limits the possible use cases. On the other hand, generic design is not as optimal from the aspect of performance, but can be used to greater extent. [69]

Layered system. In a layered system, the system has several layers that form a hierarchical structure where the higher level layers use the services provided by the lower level services. This can be used in REST to hide the actual server from clients by making them connect to an intermediary server. The intermediary server can then perform tasks such as load balancing or in cases where a legacy client makes a request it can be directed to a specific server that can handle the request as it may not be supported by newer server software. By limiting the actual server access to intermediary server only, so that the clients cannot form a direct connection to them, the layered system improves security as

tasks such as authentication and authorization can be performed at the intermediary server. [69]

Code-on-demand. Code can be seen as a resource as well. Because of this, REST allows the client to download code files, e.g. scripts, and execute them, thus enhancing the client's capabilities. This also makes it possible to make lightweight, dynamic clients that fetch the needed code for an operation from the server. [69]

While REST architecture can be implemented on top of any communication protocol, the most commonly used is HTTP. Different operations are identified by using the HTTP verbs, usually with the CRUD operations GET, POST, PUT and DELETE. The most straightforward ones of these operations are GET and DELETE: GET for fetching a resource, DELETE for deleting a resource. HTTP verb equivalent to the create operation is POST, though PUT can be also used. PUT updates the defined resource to the version sent in the request. If the resources do not exist, it will be created. Out of these four GET, PUT and DELETE are idempotent operations, meaning that multiple executions will result in the same situation as a single execution. [70] [71]

4.3 Model-View-Controller and its variants

One of the most common architectural patterns is Model-View-Controller (MVC), which is described in Figure 6. In MVC pattern, the user's interaction, whether it is a post or a simple URL, is first handled by a controller. Controller is responsible for deciding whether the interaction requires changes in the model, which then updates the view, or the controller itself can update the view if no information from the model is required. In a pure MVC architecture, model does not relay any data to the controller, but rather updates the view directly. If the user interaction results in state change in the model, the manipulation of the model is done by the controller receiving the request or input. [72]

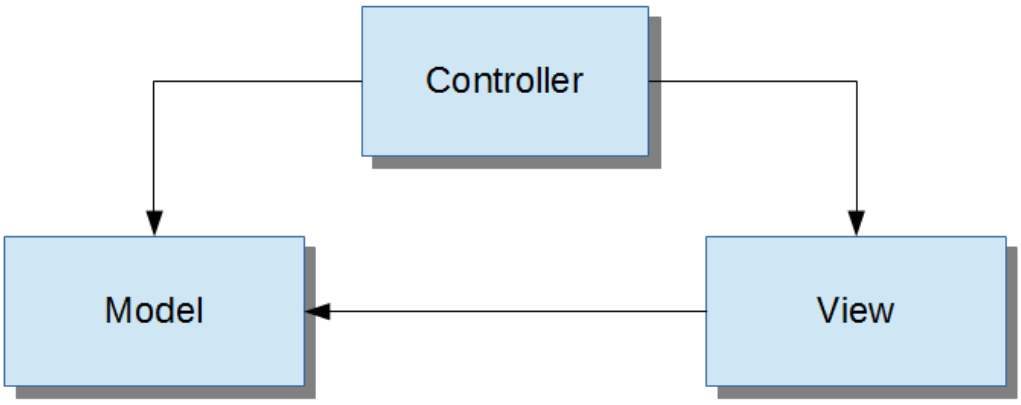

Figure 6. Model-View-Controller [72]

Model-View-Presenter (MVP) is a variant of the MVC pattern. The view is similar to the MVC's view and the presenter works much like the controller in MVC pattern. The difference between MVP and MVC is that only the presenter updates the view. Communication between the presenter and the model is partially indirect; presenter can manipulate and read the model directly, while the model sends events to notify that the state has changed. [73]

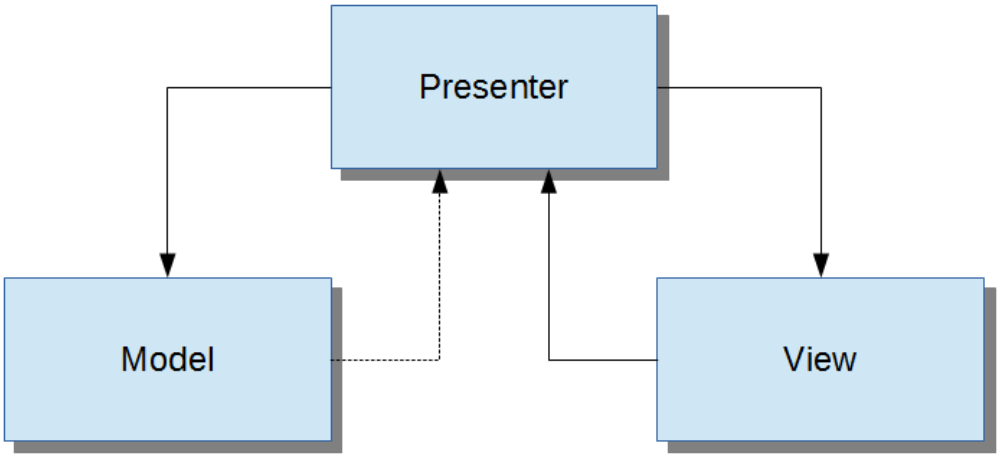

Figure 7. Model-View-Presenter [73]

Model-View-ViewModel (MVVM) is another pattern derived from MVC. It resembles closely the MVP pattern, with the difference how different components relate to each other. In MVP, the presenter is aware of the view and the model, but not the other way around. In MVVM, however, the view is aware of the view model, and the view model is aware of the model, but the relation is only to one direction. The model communicates with the view model by sending notifications, as does the view model with the view. [74]

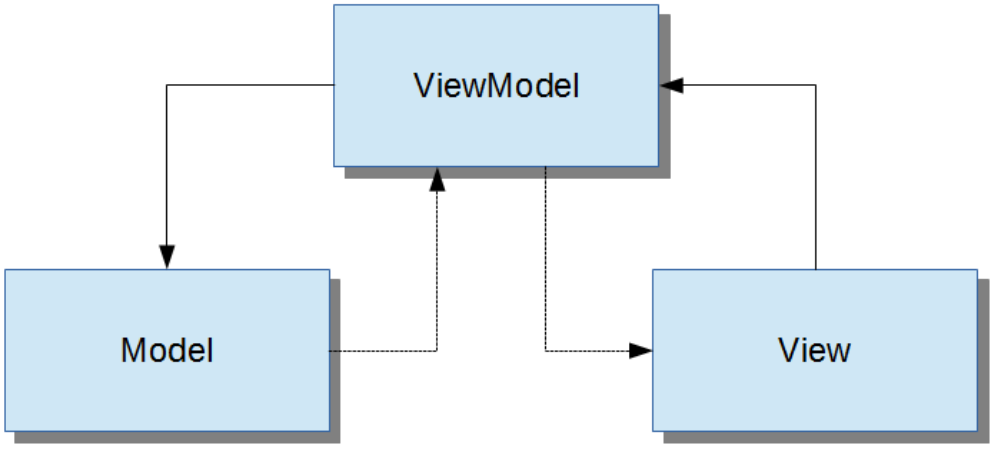

Figure 8. Model-View-ViewModel [74]

It should be noted that the descriptions and to some extent, the interpretations of the above patterns vary depending on the source. What is relevant in this work, are the components in the described patterns and how they communicate with each other. Each pattern has a view, model and a component that handles the user input and modifies the model. Common also is that the model as the data storage is not aware of the other components. Of course, in some variations of the MVC pattern, the view is aware of the model and reads data directly from there. Important in the communication is, that the model is not generally aware of the other components and the information is relayed either by model sending event or the other components reading data from the model. A third option that could be used is two-way data binding between the model and controller or corresponding component. Problem with the two-way binding is that some type of watcher is required that monitors changes in the data and does the necessary operations to it or calls functions that handle the data. Of course it should be remembered that the operations should not do alterations to the original data but rather treat is as immutable.

4.4 AngularJS

Originally developed by Misko Hevery, AngularJS is an open source JavaScript framework developed by Google and released under MIT license. It is one of the most popular Javascript frameworks for UI development. [75]

Applications developed using AngularJS are not tied to a certain architectural design pattern, but there still exists a view and some kind of model or controller. This is known as so-called Model-View-Whatever, where "Whatever" can be e.g. Controller, ViewModel or Presenter. Views in AngularJS are templates written in HTML with extended notation and non-standard elements and attributes, which are parsed and handled by the AngularJS's JavaScript when the application is ran. The template language contains also control structures like for-loop and if-statements (`else` is not defined), which makes generating HTML elements from data easy and allows to define very dynamic documents. Watchers for variables can be added in order to detect changes in them and to run code afterwards. The watchers are run in AngularJS periodically in a `$digest` loop which calls for the watcher functions until none of the listeners fire any more. While it is possible to call the digest function manually, it is not recommended. Instead, `$apply()` should be used and before the call, it should be checked that a digest loop is not running. [75] [76]

Models in AngularJS are defined by using either services or factories. Both are very similar in the way they function and both are singletons. The main difference is how they are defined, which is demonstrated in Example 3.

Example 3. Defining AngularJS models.

```
application.service('someService', function() {
    this.greet = function() {
        console.log("Hello World!");
    };
});

application.factory('someFactory', function() {
    return {
        greet: function() {
            console.log("Hello World!");
        }
    }
});
```

The function given for AngularJS' service binder is actually a class, where public methods and variables are defined using `this` keyword. With factory binder, the function returns an object where the methods and variables are defined.

AngularJS applications are hierarchical by nature. Each application has one root scope and several child scopes defined with the controllers. It is possible to access the parent scope from child but not the other way around. For communication between different controllers, there are several options. [76]

The first option is to use models to relay data. One controller manipulates the model; another one reads it. This requires of course that a suitable model is defined and contains the required functionalities, but it also makes it possible to perform operations to the data in the model. That way it is no longer necessary to do the operations in the controllers, which can lead to simpler design. As a downside, a watcher is required in the controller in order to monitor the changes to a model or part of it. Because a single watcher usually monitors a single variable or object, this might result in large amount of watcher, which again can have a negative effect on the application's performance. [76]

Another option is to use AngularJS' broadcasts. Broadcasts are events that others listen to. Compared to using a model and watchers to relay data between controllers, broadcasts are lighter but may complicate the design. If any operations need to be run for the broadcasted data, either the operation code needs to be implemented in several locations or have a separate module or modules that contain the operations. The former is not advisable as it results in codebase that is difficult to maintain and easily results in bugs and oversights. The benefit is that the broadcast listeners are called only when a broadcast is sent, unlike the watchers that are run on every `$digest` cycle. [77] [78]

AngularJS also contains directives, which can be used as an element, attribute, class or in comment. Directives can be defined to use an existing scope, or have them use an isolate scope. If an isolate scope is used, then the variables in the parent scope cannot be accessed

directly. Instead, it is defined what values can be passed to the directive and how the binding works. The data binding can be two-way, one-way (as of version 1.5), a string or a callback. It is also possible to define a separate controller for the directive. Before version 1.5 of the framework, the directives could be used in component-based design. [79]

Components were introduced in AngularJS 1.5, bringing component-based design to the first AngularJS. Before the official support, component based design was possible at least to some extent by using directives. In fact, how components are defined and used resembles very much directives. A detailed comparison of the differences of components and directives is in Table 6. [80]

Table 6: Comparison between Directive definition and Component definition [80]

	Directive	**Component**
`bindings`	No	Yes (binds to controller)
`bindToController`	Yes (default: false)	No (use bindings instead)
`compile function`	Yes	No
`controller`	Yes	Yes (default `function() {}`)
`controllerAs`	Yes (default: false)	Yes (default: `$ctrl`)
`link functions`	Yes	No
`multiElement`	Yes	No
`priority`	Yes	No
`require`	Yes	Yes
`restrict`	Yes	No (restricted to elements only)
`scope`	Yes (default: false)	No (scope is always isolate)
`template`	Yes	Yes, injectable
`templateNamespace`	Yes	No
`templateUrl`	Yes	Yes, injectable
`terminal`	Yes	No
`transclude`	Yes (default: false)	Yes (default: `false`)

5. DESIGN AND IMPLEMENTATION

This chapter goes through the application implementation. The technologies chosen for the application are HTML5, JavaScript, AngularJS, CSS and Bootstrap. For communication with the cloud back-end REST API is used in conjunction with server-sent events, which are used for notifications and live data. The data visualization library that is responsible on drawing graphs uses canvas element for rendering. Server-sent events and REST API were chosen as they were already in place in the cloud back-end, so it was only natural to use them. HTML5, AngularJS and Bootstrap were used, as they were already familiar technologies. For JavaScript and CSS there are not too many alternatives. There are languages that can be compiled to JavaScript and similar options for CSS, but it was considered not to be necessary to use them in this work.

Section 5.1 goes through the requirements set for the application and visualization. Section 5.2 introduces the end-to-end architecture and the data flow in the system. The UI and its features are presented in Section 5.3. Finally, the Section 5.4 explains the application's architecture, how the visualization library works and what features it has.

5.1 Key requirements

The application was to be designed to provide monitoring features similar to the hospital ward monitors so that it could be used or modified to function as such if needed. This section introduces some functional and nonfunctional requirements that must be met so that it can be utilized at the ward as well as a remote monitoring situation.

The default use case was set to monitoring recovering cardiac patient, from which the following measurements were assumed most relevant: electrocardiography (ECG), heart rate (HR), R-R interval (RRI), heart rate variability (HRV) and respiration rate (RR). In addition, the activity of the patient was to be followed by monitoring the posture and cumulative step count. The application should also be able to display all measurements except posture and cumulative step count as a graph. Because of this and the fact that there can be 12 ECG waves, the application must be able to display several graphs at the same time [81]. In addition, it should be possible to display several data sets in a single graph, e.g. with annotated ECG [82]. If some measurement is not displayed as a graph, the current value should be still visible as a number or text.

Both hospital ward and home monitoring scenarios require that real-time visualization is possible with the application. In the case of the hospital ward, the concept of real-time is much stricter than in the remote monitoring situation, as the response times to critical changes in patient's condition must be as low as possible. When monitoring patients remotely, for example when they are at home, it can be assumed that their situation is not

as critical as it was when hospitalized and the delay from measuring a vital sign to visualizing it can be some seconds. In this work, it is considered good enough that the end-to-end latency of a data package is approximately one second at most, preferably much lower. If the application has buffering in use and is not used in an environment with strict latency requirements, the delay between sending data package from the sensor and displaying it in the application can be 3-5 seconds.

Also in both scenarios – though more likely in home monitoring – it is required that the application is capable of displaying historical data from any time interval specified by the user. In the application, the user should then be able to scroll through the data and alter the time interval visible in graphs, each separately. While scrolling, all graphs should be synchronized to the same point of time. If there is a time marker drawn into the graph or graphs, all graphs should be synchronized to the time marked by it. The time marker also works as a reference point when the sensor's graph is hidden. Then the time indicated by it is the time by which the numerical or text value is selected for the value fields.

The goal in the user interface design is to provide good overall user experience. Therefore, the UI must respond quickly to user input and be easy to use. The user must be able to see quickly the status of all monitored patients so that they can prioritize the patients when necessary. When looking at the patient's data, the user should be able to see the patient's status at a quick glance. If the any limits are defined and exceeded, it should be able to draw the user's attention. This can be treated as a part of the alert visualization functionality, which is a required feature in the patient monitors used in hospital wards. [83] For example, this feature could be achieved with a highlight in graphs or by changing text or element background color. As the application is a demonstration whether web technologies can be used in this context, audial alerts are not required.

It should be also noted that there are standards defined that patient monitors should follow. Some of these standards concern technical aspects of the devices, but some, such as ANSI/AAMI EC13 are specifically for displaying the data [83]. Because of the scope and nature of this work, these standards are not followed. Instead, filling up the terms set by different standards are left for possible future development.

5.2 End-to-end architecture overview

Figure 9 illustrates the flow of data from end to end. Sensors measure the data from a patient and send it to the gateway device that is capable to communicate over the Internet. The data is sent inside a JSON document to the inbound server using the REST API it provides. Inbound server relays the data to the cloud, which stores it and if available, can perform real time analysis to the data, creating automated annotations, events and alerts based on the data. The cloud also sends the data and possible annotations to the user interface using server-sent events. The UI is loaded through the outbound server that provides the REST API for outgoing data. The REST API is also used when registering to

listen the server-sent events. Although inbound and outbound servers are separate in the image, it is natural that they can be combined to a single server. However, separate instances and servers provide some redundancy and fault tolerance that is needed.

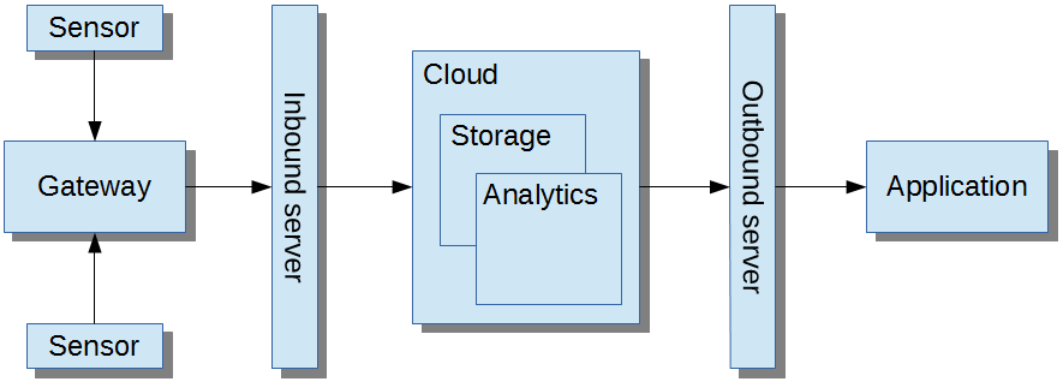

Figure 9. High-level end-to-end architecture

The communication between the cloud back-end and the web application is currently mostly unidirectional. Because of this, the server-sent events can be used for the data flow from the back-end to the application, while data requests can use the REST API the server provides. At least for now this is enough and the overhead caused by the HTTP protocol is acceptable, but if the requirement arises that, there should be more two-way data flow with less overhead, using WebSockets could be considered.

5.3 Overview of the implementation

The web application consists currently of two views: patient list and patient data. Common for both views is the top bar, which shows the current location, notification list button, and the name and icon of the user logged in. In addition, the bar contains an arrow button which allows the user go back to the previous view. Though working the same way as the browser's back-button, it is needed in case the UI is used through a web-view component where there might not be browser's implementation for it. Notification list button also shows if new unacknowledged alerts have arrived since the session start. Clicking the button will open a list of alerts which lists all alerts arrived during the session in two categories depending on whether they are acknowledged or not.

In Figure 10 is visible the default view of the application: the patient list. It shows the current situation of the patients under doctor's or nurse's care, or if a patient is looking his or her own data, then only the patient in question. For each patient, information on alerts raised by the system and the status of the sensors and device battery is shown. Alert information consists of the date and time of the most recent alert and how many alerts there has been in the last 24-hours, shown by both type and the total amount. This is to give the

doctors and nurses a quick way to assess the situation of a patient and to see if the device is working as intended and if it is connected. The patients can be organized by the amount of alerts during the last 24-hours, alphabetically or by the latest alert. For sorting by alerts, only descending order is supported and for alphabetical sorting ascending order is used.

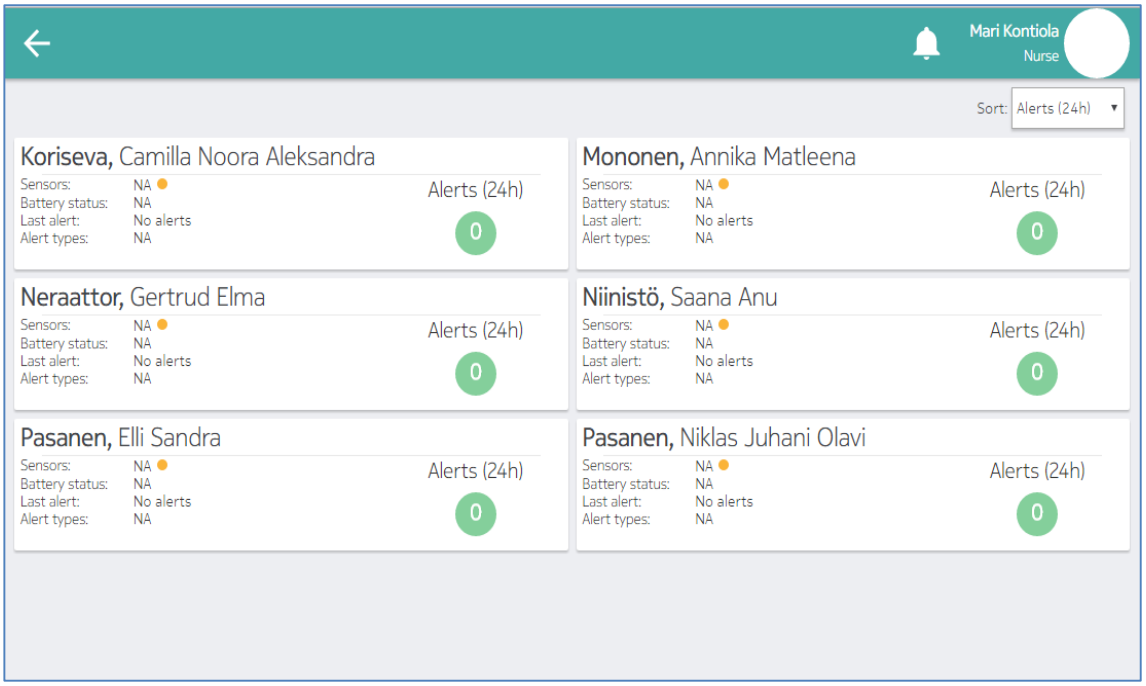

Figure 10. Patient list view.

Clicking a patient in the patient list view takes the user to the patient data view shown in Figure 11. In the figure, only ECG is currently visible as a graph. For other values, a numerical value or describing text is shown. In the graph, the plus and minus buttons are for increasing and decreasing the time interval from which data is displayed in the graph. By clicking the cross, the graph can be closed after which the sensor is shown just like the others in the figure. The graph can be scrolled by dragging it right and left. Above the graph are its controls. From them it is possible to skip to the previous and next alert, skip one minute backwards and forwards and to display historical data by selecting a time range from calendar. At the moment of taking the screenshot, the last and next alert buttons were disabled. What is not visible in the screenshot is the “real time” button, which would continue rendering the data coming from the server if the user has stopped the real time rendering by dragging the graph. The time marker is also not visible in the figure. It appears into the graph under the time when the mode switches from the real time.

Below the graphs are rest of the sensor values. The sensors in solid circles can be opened up as a graph by clicking the respective “Open graph” button, while the hollow ones only have the current value visible. The value in the circle corresponds to that sensors value at the time indicated between the graph control buttons which shows the time of the graph’s right edge when in real time mode and otherwise the time at the time marker’s position.

Below the sensor values are the annotations, or alerts and events. They are organized by the date with the latest one at the top. If the entry in the list is an alert, then a red border is added to it. Clicking an alert or event centers the graph and to the time of it.

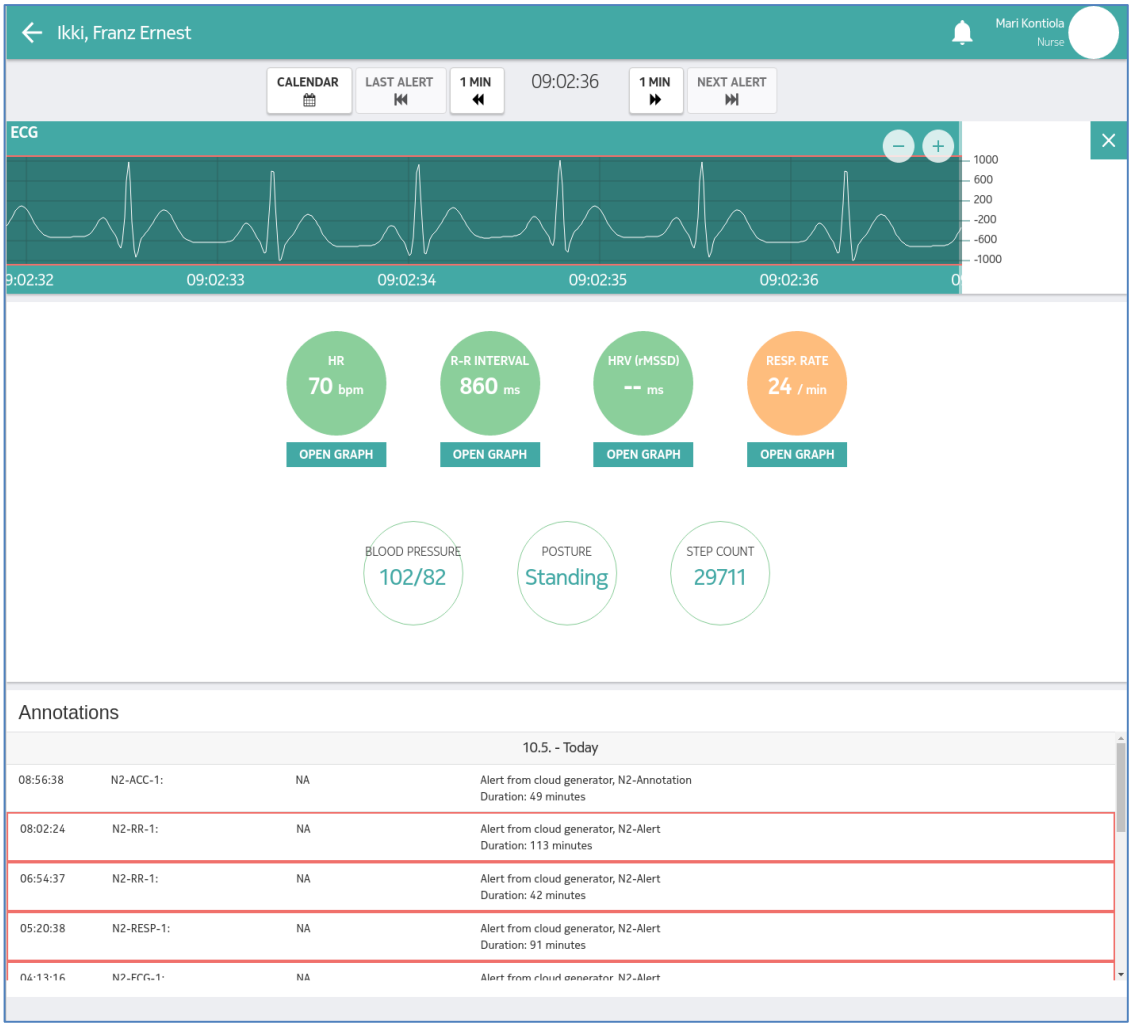

Figure 11. Patient data view.

Nearly all changes in the view caused by the actions of the user are reflected to the URL. Only navigation in time does not change the URL, but everything else, such as opening a graph, closing it or selecting a time range from which to show data, changes it. This way it is possible to create hyperlinks to time intervals and have them open the right sensors as graphs when arriving to the view. In the Table 7 are the supported query string parameters.

Table 7. Patient data view query string parameters

Field name	Description
`show`	[OPTIONAL] Tells what data is shown as graph in the view. Can be given several times to open multiple graphs at once.
`start`	[OPTIONAL] Unix epoch indicating the start of the time interval. Defaults to end – 3 600 000ms if end is defined.
`end`	[OPTIONAL] Unix epoch indicating the end of the time internal. Defaults to start + 3 600 000ms if start is defined.
`id`	Patient identifier that determines which patient's data is shown.

5.4 Application implementation

AngularJS encourages the programmer to choose what is the most suitable architecture for the application case by case. In this application, an architecture that is closest to the MVVM pattern described in Section 4.3 is used. In order to properly evaluate the suitability of canvas for rendering data, a custom visualization library was written for the application. Subsection 5.4.1 describes the dataflow and the architecture of the implementation, while subsections 5.4.2 and 5.4.3 concentrate on the visualization library implementation and its features.

5.4.1 Architecture

Because AngularJS does not force one to use any specific design pattern, it gives the programmer freedom to design the optimal architecture for each application. It also allows combining different patterns and architectural styles if needed. The application is closes to MVVM pattern, but takes influence on other patterns as well.

In Figure 12, the application’s architecture, data flow and different components are described. In the application, the changes to the models occur either when the user clicks a link or when server-sent event is handled. In the first case, the link is either an URL containing a query string or a function call from AngularJS template with the necessary information in order to show the right view with the right content. Whether using an URL or a template function call, the interaction goes through a controller, which then can load more data from a server and update the model with the data received. In the latter case, the model is modified by the server-sent event listener service without the controller being an intermediary.

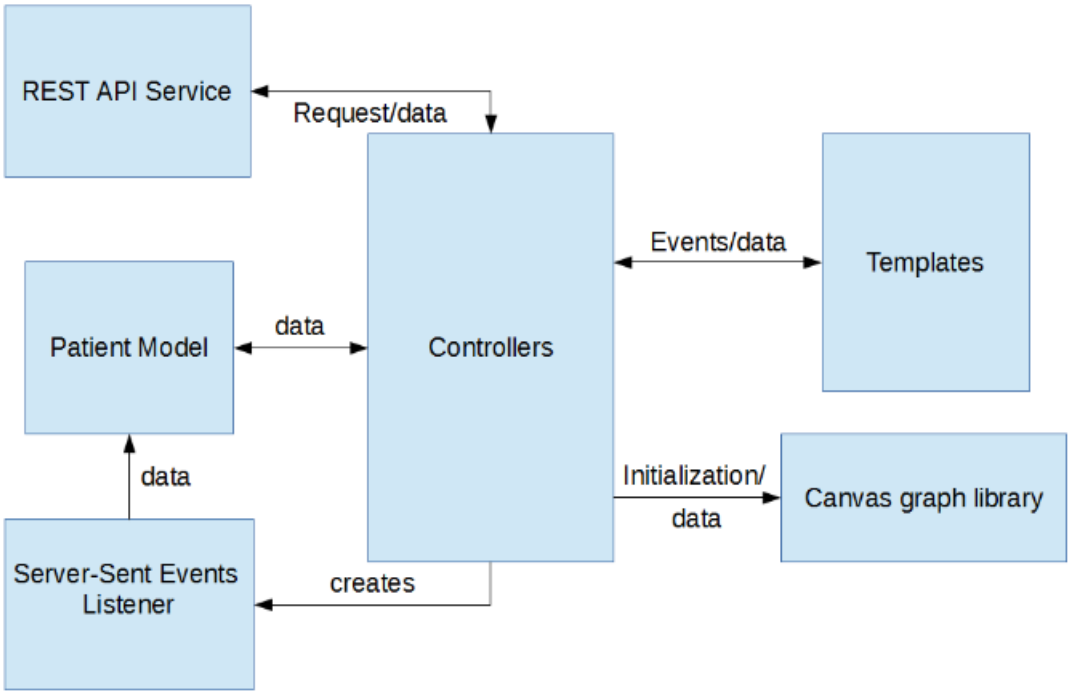

Figure 12: Application architecture

Services and models. The services and models are responsible for fetching and storing the data. Inside the patient data model is contained all the received information about the patients including the sensor data. The model provides an API through which the other components can manipulate and fetch data. The SSE listeners are started by the application and take care of receiving the data coming from the cloud via server-sent events and manipulating the patient data model accordingly. REST API communication service is an abstraction layer for making calls to the REST API provided by the cloud. The data returned in the response is relayed directly to the caller, which then can for example manipulate the data and push it into the patient data model. The data flow and connections between different components are visualized in Figure 12.

Controllers. The controllers contain the state of the view or partial view, take care of responding to user input and choose what data is relayed to the view. Application level controller initializes the application and takes care of some top-level functionalities. Other controllers are responsible for a single view or part of it. Communication between the controllers, services and models is done partially through API calls and partially through references to the data. E.g. after a reference to patient's data is returned to a controller from the model, the controller can define an AngularJS watcher for it which is responsible for detecting changes in the data. By doing this, it is possible to give the visualization library a reference to the visualization library and still have all the data in single location.

Templates. The templates do not contain any business-logic, but instead are responsible for describing the view or portion of it. The language used in the templates is a mix of

HTML5 and AngularJS' own template language. Views are updated by using `$scope` variables, which allows values in the UI to be easily changed.

5.4.2 Canvas visualization library

The canvas visualizer is a custom component written for the application. The basic idea in the library is that it does not take the ownership of the data or copy it to its internal data structures. Instead, the component takes a reference to the data that is in the required data format. The dataset can be given limited amount of styles, such as whether it should be drawn either as a line, a square line or as an area highlight. Each data set is also given a unique name that can be later used to identify them and for example drop to remove and replace a certain set. The library uses a time window to define what data should be drawn.

Using time window and a binary search to look up the starting index was the only real choice for the library. Other solution would be to use indexes, but it would introduce considerable problems and might even limit the feature set. With time window, it is easy to have several data sets in one canvas, data sets are not limited to a constant sampling rate and optimizing the performance by dropping older data does not require additional code in the library. When plotting data in any use case mentioned in chapter 5.1, drawing to the canvas works in similar fashion. The time window is defined by setting the end time and time interval length. The first can be set explicitly and implicitly; if it has not been given, then it defaults to the earliest element of all data sets given to the library. With live charts, the initial end time can either be given or left for the library to deduce, but in case of static data, it needs to be set explicitly.

In live visualization, the animation works by moving the end time forward the amount of time between two frames. The time added depends on the framerate, which in turn is dependent on what framerate does the browser provide and how long time interval is shown on the graph. The former is because the library uses `requestAnimationFrame()` [84]. It aims to provide 60 callbacks per second, but may vary depending on the refresh rate provided by the display. In the application, the framerate is not only dependent on the browser, but also on the length of time interval visible on the graph. In live visualization, the time between the frames can be increased to match the time interval that one pixel represents on the canvas. This way showing a large amount of data on live canvas should be relatively light task.

In addition to data, time markers and horizontal lines are drawn on the canvas in order to fulfill the basic user requirements. Time markers are drawn with variable intervals depending on the length of the graph's time range. The horizontal lines give the user a basic idea of the values on the graph. They also do not have a constant distance from each other, but instead are dependent on the graphs value range. The algorithm responsible on calculating and drawing the horizontal lines aims to adjust them so that the zero level should be visible. This depends on the minimum and maximum value on the current period. If

the minimum is above zero or the maximum below, then it is not guaranteed that the zero line is drawn. One noticeable difference between time markers and the horizontal lines is that the horizontal lines are redrawn only when necessary and the drawing is done to a different canvas. This is an optimization to reduce the amount of drawing done to a single canvas on a single refresh cycle.

In the application, it is necessary to show either the graph's end time or the middle line's time depending on whether the canvas is in static mode or live mode. The time is sent as a JavaScript event, where the document is the target. This way any component interested in the time can add an `EventListener` to catch the events. Live graph sends the end time in every `requestAnimationFrame()` cycle, while a static graph showing either received or retrieved data sends the time represented by the middle line.

5.4.3 Visualization features

The features for the visualization are mostly from the hospital ward scenario, where the patient is monitored closely and there are several different vital signs visualized and monitored at the same time. The implemented visualization provides the following functionalities:

- Drawing on several canvas elements at the same time both synchronously and asynchronously
- Multiple data sets on one canvas
- Tooltip
- Scrolling
- Changing the length of time shown on the canvas
- Different types of curves are line, square and area highlight
- Real time visualizing
- Static data visualization

Because the application must be able to display several live graphs at the same time, the visualization library must have support for drawing on several canvas elements at once. In addition to having a synchronized multi-canvas support, asynchronous drawing was also implemented, though it was not required. With asynchronous drawing, it is possible to display different period on each graph and have them work separately from each other, while having a single instance of the library.

The visualization library can be set to two different state: static and real-time. This is determined when creating the instance and for the state to change, a page reload is required. In static mode, a set of data is given to the library to show. In this context, the data in this situation is historical data that the user wants to look at. When the visualization library works in static mode, only animation is disabled. In real-time mode, the library can also receive a set of data, but it also starts to play it from the beginning. Adding data

to the end of the data set makes it possible to monitor the patient in real-time. In both modes, data can be added or removed from the data set as it is governed by the component that owns the data. A reference to the data is given to the library but it neither claims the ownership of the data nor copies it or stores otherwise.

During the development, it became obvious that there has to be support for different types of graphs. Better readability can be achieved by rendering measurements such as respiration rate and R-R interval differently from ECG. For ECG line-type is used, because a direct line from one measurement to another can be drawn. However, RRI and respiration rate use square-type, where one measurement is valid until the next one. It can be thought that a data point has a duration in this case and throughout the duration, the one value is valid. This results in graph that has step-like changes instead of linear. The area highlight is a special type. As can be deducted, it is used to highlight a rectangular region. This is used only for alerts currently and does not have any customization options.

Related closely to the area highlights is the possibility to have several data sets on a single canvas, because highlights are defined as data sets. Because it must be possible to show both the area highlight as well as the actual data, the visualization library supports multiple data sets on a single canvas element.

Other basic features offered by the library are tooltips, scrolling and changing the length of time shown on the graph. Tooltips were added to provide additional usability and to provide a way to see the measured value at a specific time. Because it is possible show historical data, there must be a way to scroll the data and thus, scrolling was added. It is also possible to change the length of time visible, or “zoom” the graph, though it only changes the scale on x-axis. This is particularly useful when looking at sensor data such as respiration rate or blood pressure, where the trend is often needed.

6. EVALUATION AND LESSONS LEARNED

In this chapter, the performance and technological choices are evaluated. Because the application was done for demonstration purposes and the main idea of this thesis is to study if web technologies can be used in this context, the user experience and usability evaluation is not a priority. Additionally, the evaluation would require feedback from the actual users, more precisely from doctors and nurses. Unfortunately, this was not possible and instead, it is left for future work. Despite of this, it can be said that the UI does not currently fulfill the requirement that the user should be able to see the patient's current situation at a quick glance. This is because there is no clear indication of alerts other than the alert and event list. Both it and the highlight in the graph are too subtle to draw attention. The performance is evaluated by observing the performance in the development environment with Google Chrome browser running in configuration described in Table 8.

Table 8. Specifications of the computer used in development and performance testing.

Processor	Intel i7-5600U, running at 2.6 GHz
RAM	8GB DDR 3
Graphics	Intel HD Graphics 5500
Operating System	Gentoo Linux (kernel 4.1.15-gentoo-r1) with K Desktop Environment (KDE)

The application performance evaluation was done by measuring central processing unit (CPU) and memory usage and by observing the behavior of the application when used. Measuring the values was done by observing them in KDE System Guard process list, as it was not possible to get process specific graph and statistic. In the technological evaluation most of the end-to-end chain is taken into account and contains assessment of end-to-end latency, suitability of the technological choices and development experience.

6.1 Application performance

The overall performance of the application varies from good to average and is dependent on the situation. When used with the data, alerts and events mocked, the application performs well when first loaded. However, in longer use the performance degrades. The main reason for this is not the amount of data stored locally, but the way AngularJS updates the elements created in template for-loops: all elements are re-rendered, instead of han-

dling only the changes. In the patient list, this does not cause any visible issues in performance. However, in patient data view with live data, as the amount of received alerts and events becomes large enough, updating the lists can cause visible interruptions in graphs. Reason for this is that if the event or alert is for the patient whose data is being monitored, two lists needs are updated. How likely issue this would be in a real use, is unknown as most of the testing and performance evaluation is done with mocked data. Further evaluation of this would require deeper knowledge of the medical domain.

Visualization of the live data is responsive and able to handle several live graphs at the same time. By default, ECG data is shown as a graph when the user arrives to the patient data view and for other sensors, the value corresponding to the right edge of the graph is shown as a value. The number values are updated approximately every 200ms and thus does not cause much CPU load. The live graph is the most demanding visualization as it is rendered 60 frames per second at most and can contain from 3 seconds to 10 minutes of data. In the case of ECG with 100Hz sampling rate, this means there can be from 300 to 60 000 data points in the graph. With the optimizations in the canvas visualization code, the CPU load is at most 9-12% and decreases once the framerate can be dropped. When several graphs are rendered, the CPU load of the Chrome process can exceed 30%. Figure 13 contains the CPU usage measurements taken using Monitorix [85] when the application was running. The baseline is from the start of the graph to approximately 15:34. During this, no graphs were active, but the application was running and displaying the incoming data as values. The baseline CPU usage was approximately 1%. After measuring the baseline, one graph was activated. When one graph was active, the total CPU usage of Chrome was between 14-16%. Around 15:57 three more graphs were activated resulting in CPU usage between 34-38%. Considering the fact that the measurements were done in a high-end laptop, it is very likely that the performance is not as good in the current low to medium range computers.

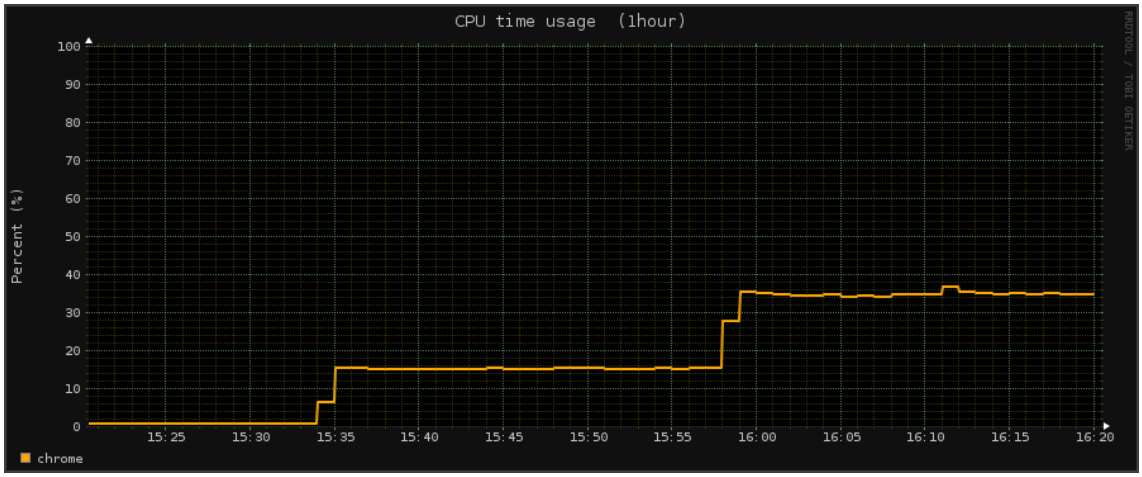

Figure 13. One hour CPU usage benchmark in Linux.

The performance when scrolling the data depends greatly on how long interval is visible and on how many data points it contains. When the interval is limited to the maximum of

10 minutes, scrolling works well enough even when showing the maximum time interval. If the limitation is disabled, the performance begins to degrade, as there are no similar optimizations in place as there is for the live graph.

As a comparison, the application was also run in Windows 10 environment. The configuration for the computer is in Table 9.

Table 9. Windows test configuration

Processor	Intel i5-3570K (Ivy Bridge), overclocked to run at maximum of 4.8GHz
Memory	24 GB DDR3
Graphics	ASUS GeForce GTX 670
Operating System	Microsoft Windows 10

During the testing in Windows environment it was noticed that the CPU usage of the application depends greatly on the test environment. During the test, the combined CPU usage of the two Chrome processes used in running the application was most of the time between 10-12% when rendering to four canvases at the same time. Even if the desktop configuration is more powerful than the laptop configuration used in development, the CPU never reached the full speed of 4.8 GHz during the test. Instead, it varied most of the time between 1.9 GHz and 2.6 GHz, occasionally peaking briefly close to 4.0 GHz. The occasional peaks were considered anomalies caused by some other software running on the computer at the same time. One likely reason for the lower CPU usage is canvas graphics processing unit (GPU) acceleration. GPU usage increased noticeably from the baseline when the application was rendering the canvases as visible in Figure 14 and Figure 15. The two peaks in Figure 15 were caused by opening and closing the fifth canvas graph.

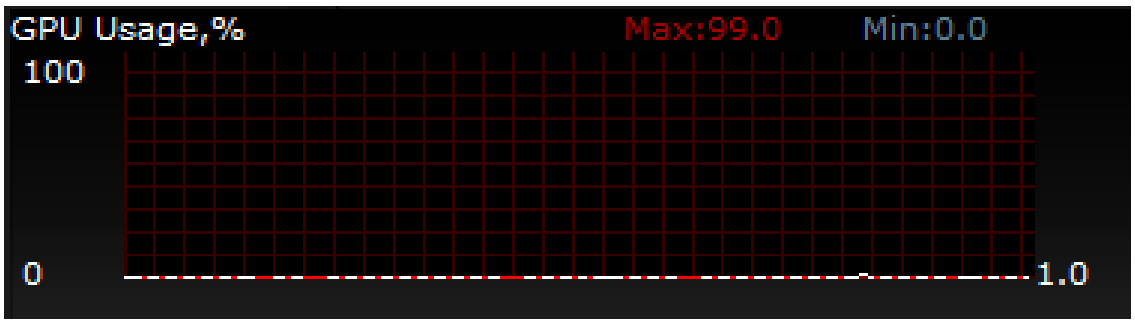

Figure 14: Comparison test GPU baseline

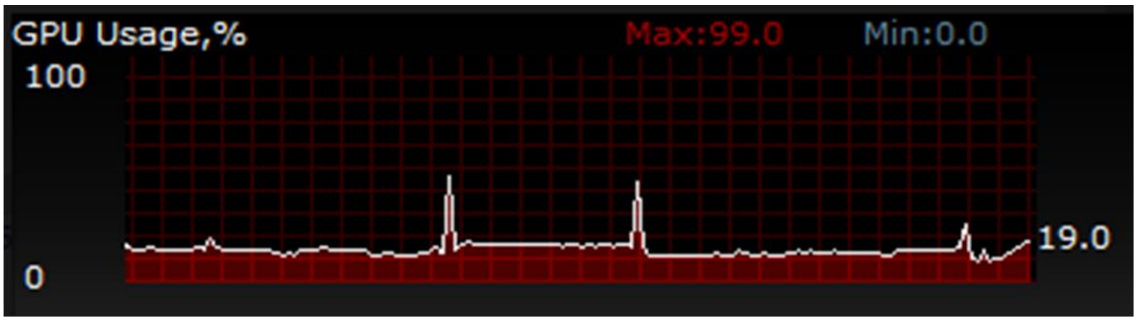

Figure 15: Comparison test GPU usage during the rendering

The minimum and maximum values visible in the figures are not applicable as they are collected from a longer period and during that, the GPU has been in use. For the GPU usage measurements, ASUS GPU Tweak software was used. During the testing, the GPU clock speed stayed mostly at the minimum of 324MHz.

In real time monitoring, the application was never used long enough for it to store locally longer periods of data and memory consumption never became an issue. At the start, the memory consumption of the Chrome process was observed to be between 70MB to 100MB. As more data was received, the memory consumption steadily increased and occasionally dropped as Chrome garbage collection released some of the reserved memory. As an extreme test, a maximum amount of data that the server could handle at one time was requested. Because of technical limitations, the server could return 5-6 hours of data at most. Once the data was fetched and transformed to the internal format, the memory consumption of the application was between 210MB and 250MB, which is acceptable on PC.

The application was also briefly tested with NVIDIA Shield tablet, where real time monitoring was considered to work well when showing several live graphs. Scrolling was only tested on a single graph and it was considered to work well. Memory consumption was not tested, but the maximum data request test was performed with the tablet as well. Most of the time the tablet browser could not handle the large amount of data and the test failed as the Chrome process crashed. The exact reason for the crash, however, is unknown and looking into it was considered low on priority.

6.2 Technological evaluation

The chosen technologies performed well enough in the scope of this work to be considered usable in real situation. The generic IoT back-end worked reasonably well, the end-to-end latency being most of the time relatively low.

In a test, 3200 packages of mocked data were sent to the cloud at the rate of one package per second. Each package contained a timestamp at millisecond precision that was added to it when the package was created. On the same machine, a server-sent event listener

running in Chrome Developer Tools' console was comparing the timestamp in the package to the current time. The difference between these two timestamps was calculated and stored and after collecting enough samples, average latency was calculated. In addition, minimum and maximum latencies were sought from the gathered data. The average latency was found to be approximately 335ms, minimum being 295ms and maximum 5318ms. Only three samples exceeded 1000ms latency, however, during the development the over five second delay was noticed to occur more often than during the test. Because of this, the current cloud back-end does not fulfill the requirements of hospital ward monitoring, as it requires steady and low latency.

Aside the performance issues, AngularJS was otherwise a good choice as a framework. The MVW pattern made it easy to split the program code naturally and everything required was provided under a single framework, though still split to different modules so that only required functionalities could be included. However, during the development process it was noticed that some features are either hard to do or nearly impossible. One such feature was creating a single table with day separator rows between alerts or events of different days. It may be possible to solve most or all issues that came up by using the components introduced in AngularJS 1.5, but without extensive refactoring mixing component based design with the MVW architecture can make the overall architecture more complicated and harder to maintain. When compared to, for example React [86], another increasingly popular framework, the template language of AngularJS does not compare well. Because the control statements are defined inside HTML elements as attributes, they make the readability and understandability of the templates harder.

Bootstrap framework was chosen pre-emptively to cater the possible upcoming needs. However, it was mostly used for the grid layout and the alert drop-down box to provide easy responsive design and thus most features of it were left unused. As only the layout features are used, having the whole Bootstrap in the project is too heavy and likely adds overhead to the application. Instead of using Bootstrap for the layout and the drop-down box, alternatives that provide only the required features should be used.

Considering performance, Canvas worked relatively well. From the point of view of development, the canvas was moderately difficult to work with. Although the actual drawing to the canvas was considered to be easy, controlling what to draw and how was moderately difficult. Part of the difficulty came from finding the lightest way to draw to the canvas and trying to avoid unnecessary canvas state changes. Comparison to SVG cannot be done at this point since there is no experience of drawing with JavaScript to SVG element.

The proprietary data format used to transmit data from end-to-end worked well. While JSON might be problematic if the gateway device has very little memory, in this case it was not an issue. More optimized way to transmit data from the gateway device to cloud would be to use binary format. The use of proprietary format provided a good way to

have only the required fields and values in a data package with acceptable overhead. While the data formats introduced in Section 2.4 provide more complete way to store and transfer larger data samples, they were too big specifications or otherwise unsuitable for the application. For example, the issue with EDF+ is it being binary, while for REST and JavaScript applications JSON is more natural. If data were to be transferred from system to another, EDF+ and FHIR are both reasonable choices.

6.3 Future development

The application described in this work provides a good basis and a point of comparison for future work. Both functional and non-functional aspects need to be addressed if the application is developed further.

Because the development team did note have access to medical professionals during the development process, evaluation of the user interface and the visualizations could not be done. Therefore, a study should be organized if any further development is done, to validate the design. This study can have a great impact on the application and the user interface and it is advisable to be done before the next version is designed and specified. Currently some features are already planned based on the assumptions made of what the end users would likely need. First feature is data loading when scrolling the canvas. If the graph is scrolled to a period that data is not loaded yet, it should trigger a data fetch. Another feature is a drilldown view for alerts and events, so that clicking one would show detailed information and possibly vital signs for the duration specified in the event.

As the performance evaluations shows, there are some issues in the application that require closer look. The first one of these is the chosen framework. Even if AngularJS performed well enough at this point of development, it is apparent that it does have some limitations and performance issues. In addition, the web development is slowly moving away from frameworks like the first AngularJS to frameworks that better support component based design pattern. Though it is possible to mimic component design by dividing the larger controllers to even smaller and smaller ones, or use AngularJS 1.5's components, the age of the framework is starting to show and the components will not be as clean and easily maintainable as with for example React. AngularJS 2 improves the syntax and brings the framework closer to React, but the historical burden is visible, especially in the template language, which can be problematic at times. The possible frameworks to replace the first AngularJS are currently React and AngularJS 2, but others such as CycleJS [87] or Ember [88] should be evaluated before making the final decision.

In further development, the performance comparison of rendering to canvas and SVG elements should be done, as the canvas was chosen with only a superficial look into SVG. This comparison should take into account the performance of a custom written visualization library as well as different existing libraries. A custom written library can be easily modified to fit the needs of the application, but existing libraries such as D3 [89], NVD3

[90] or HighCharts [91] have many problems already solved and support can be provided for them. In the comparison of drawing to canvas and SVG elements, the GPU acceleration should also be taken into account, mostly whether it should be trusted that it is available and if it is used by both.

7. CONCLUSIONS

The goal of this thesis was to design and implement an application for patient remote monitoring and to assess if modern web technologies can be used to this purpose. Some of the chosen technologies came from the existing cloud IoT implementation and the rest were chosen either by their familiarity or estimated suitability. In the design the overall goal was to make monitoring as easy as possible, helping the health care professionals in their work and

Because of the scope of the thesis, the evaluation was limited to performance and overall technological evaluation, leaving usability and UI evaluation outside. During the performance evaluation, it was noticed that some problems exist and in some extreme cases, the application does not perform well. In addition, the performance is very dependent on the environment and setup in which the application is used. On one hand, this is understandable and has to be accepted, but on the other hand, either the hardware specification for which the application is developed to be locked, or optimize further the application. Of course, the option that both options are done. In general, the performance can be considered good and the technological choices were successful.

For better maintainability and to solve certain performance issues, switching from AngularJS to some other framework could be considered and even recommended. This would require evaluation of available front-end frameworks. The technological choices were otherwise successful.

Based on this thesis, it can be said that the chosen web technologies can be used for remote monitoring, though some optimizations are required. Because of this, it is recommended that further study should be done that compares the technologies of this work to other options. If the application is developed further, a critical part of the work is to evaluate the UI and to gather input and domain knowledge from health care professionals. Additionally, if the application is to be used in real situations, it must be made sure that it conforms to the standards used in this field. In addition, security concerns must be addressed as the end-to-end system contains personal information about the patients.

BIBLIOGRAPHY

[1] OECD.Stat, Organisation for Economic Co-operation and Development, website. Available (accessed on 10.5.2016): http:// stats.oecd.org/

[2] G. I. Barbash, S. A. Glied, New Technology and Heath Care Costs – The Case of Robot Assisted Surgery, The New England Journal of Medicine, Vol. 363, No. 8, 2010, pp. 701-704. Available (accessed 11.5.2016): http://www.nejm.org/doi/full/10.1056/NEJMp1006602

[3] B. Chaudhry, J. Wang, S. Wu, M. Maglione, W. Mojica, E. Roth, S. C. Morton, and P. G. Shekelle, Systematic Review: Impact of Health Information Technology on Quality, Efficiency, and Costs of Medical Care, Annals of Internal Medicine, Vol. 144, No. 10, 2006, pp. 742-752. Available (accessed on 11.5.2016): http://annals.org/data/Journals/AIM/20115/0000605-200605160-00125.pdf

[4] M. B. Buntin, M. F. Burke, M. C. Hoaflin and D. Blumenthal, The Benefits Of Health Information Technology: A Review Of The Recent Literature Shows Predominantly Positive Results, Health Affairs, Vol. 30, No. 3, 2011, pp. 464-471. Available (accessed on 11.5.2016): http://annals.org/data/Journals/AIM/20115/0000605-200605160-00125.pdf

[5] Vital Signs: MedlinePlus Medical Encyclopedia, U.S. National Library of Medicine, webpage. Available (accessed on 12.5.2016): https://www.nlm.nih.gov/medlineplus/ency/article/002341.htm

[6] Philips IntelliVue MX800 Patient Monitor Technical Data Sheet. Available (accessed on 11.5.2016): http://www.achats-publics.fr/MEDICAL/Dm-new/Monitorage/multidisciplinaire/PHILIPS/MX/Tech-MX800.pdf

[7] GE Healthcare B40 Patient Monitor Brochure. Available (accessed on 11.5.2016): http://www3.gehealthcare.com/~/media/documents/us-global/products/patient-monitoring/brochures/b40%20patient%20monitor/gehc-brochure-b40-patient-monitor.pdf?Parent=%7BEF03B7A1-2EFC-426E-85C8-3265CF9C5699%7D

[8] GE Healthcare B40 Patient Monitor – Clinical Measurements & Modules, webpage. Available (accessed on 11.5.2016): http://www3.gehealthcare.com/en/products/categories/patient_monitoring/patient_monitors/b40_patient_monitor

[9] Fenno Medical Central Monitor CNS-6201 brochure. Available (accessed on 14.2.2016): http://www.fennomedical.fi/files/fennomedical/Potilasvalvonta/Central_monitor_cns6201_en_1107_s.pdf

[10] E. Madigan, B. J. Schmotzer, C. J. Struk, C. M. DiCarlo, G. Kikano, I. L Piña, R. S. Boxer, Home Health Care With Telemonitoring Improves Health Status for Older Adults with Heart Failure, Home Health Care services Quarterly, Vol. 32, Issue 1, 2013. Available (accessed on 13.2.2016): http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4002284/pdf/nihms566559.pdf

[11] U. Koivisto, P. Raatikainen, Rytmihäiriötahdistinpotilaiden etäseuranta, Sydänääni, No. 1A, 2011. Available (accessed on 23.5.2016): http://fincardio-fi-bin.directo.fi/@Bin/62911fbcda2c8d94b7fae934239027b7/1464026418/application/pdf/1135460/sa_teema1A_11_luku11.pdf

[12] Readmissions Reduction Program (HRRP), Centers for Medicare & Medicaid Services, webpage. Available (accessed on 10.5.2016): https://www.cms.gov/medicare/medicare-fee-for-service-payment/acuteinpatientpps/readmissions-reduction-program.html

[13] D. A. M. J. Theuns, J. C. J. Res, L. J. Jordaens, Home monitoring in ICD therapy: future perspectives, Europace, Vol. 5, Issue 2, 2003, pp. 139-142. Available (accessed on 16.2.2016): http://europace.oxfordjournals.org/content/europace/5/2/139.full.pdf

[14] G. H. Crossley, J. Chen, W. Choucair, T. J. Cohen, D. C. Gohn, W. B. Johnson, E. E. Kennedy, L. R. Mongeon, G. A. Serwer, H. Qiao, B. L. Wilkoff, Clinical Benefits of Remote Versus Transtelephonic Monitoring of Implanted Pacemakers, Journal of the American College of Cardiology, Vol 53, Issue 22, 2009, pp. 2012-2019. Available (accessed on 10.2.2012): http://www.sciencedirect.com/science/article/pii/S0735109709031969

[15] N. Varma, A. E. Epstein, A. Irimpen, R. Schweikert, C. Love, Efficacy and Safety of Automatic Remote Monitoring for Implantable Cardioverter-Defibrillator Follow-Up, Circulation, Vol. 122, 2010, pp. 325-332. Available (accessed on 15.5.2016): http://circ.ahajournals.org/content/122/4/325.full.pdf+html

[16] Silvermedia – Innovation IT, website. Available (accessed on 10.2.2016): http://silvermedia.eu/

[17] Comarch Healthcare, website. Available (accessed on 10.2.2016): http://www.comarch.com/healthcare/

[18] Medixine – Terveydenhuollon sähköiset ratkaisut, website. Available (accessed on: 10.2.2016): http://www.medixine.fi/

[19] Medtronic, website. Available (accessed on 10.2.2016): http://www.medtronic.com/

[20] ZephyrLIFE Home Remote Patient Monitoring, Medtronic, webpage. Available (accessed on: 10.2.2016): http://www.medtronic.com/covidien/products/health-informatics-and-monitoring/zephyr-life-home-remote-patient-monitoring

[21] ZephyrLIFE Remote Patient Monitoring Sell Sheet, Medtronic. Available (accessed on 10.2.2016): http://www.medtronic.com/content/dam/covidien/library/us/en/product/health-informatics-and-monitoring/zephyr-home-product-details.pdf

[22] Formats – FHIR v1.0.2, Health Level Seven. Available (accessed on 14.2.2016): https://www.hl7.org/fhir/formats.html

[23] J. D. Trigo, Á. Alesanco, I. Martínez, J. García, A Review on Digital ECG Formats and the Relationships Between Them, IEEE Transactions on Information Technology in Biomedicine, Vol. 16, Issue 3, 2012, pp. 432-444. Available (accessed on 28.11.2015): http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6086625

[24] European Data Format (EDF), webpage. Available (accessed on 19.12.2015): http://www.edfplus.info/index.html

[25] EDF+ specification, European Data Format, webpage. Available (accessed on 18.5.2016) http://www.edfplus.info/specs/edfplus.html

[26] HL7 Standards – Master Grid, Health Level Seven, webpage. Available (accessed on 10.5.2016): http://www.hl7.org/implement/standards/product_matrix.cfm

[27] HL7, Kanta, webpage. Available (accessed on 20.12.2015): http://www.kanta.fi/web/ammattilaisille/hl7

[28] Resource – FHIR v1.0.2, Health Level Seven, webpage. Available (accessed on 9.5.2016): http://www.hl7.org/fhir/resource.html

[29] Narrative – FHIR v1.0.2, Health Level Seven, webpage. Available (accessed on 9.5.2016): http://www.hl7.org/fhir/narrative.html

[30] FHIR Overview – Developers, Health Level Seven, Health Level Seven, webpage. Available (accessed on 9.5.2016): http://www.hl7.org/fhir/overview-dev.html

[31] Resource Index, Health Level Seven, webpage. Available (accessed on 9.5.2016): https://www.hl7.org/fhir/resourcelist.html

[32] FHIR Timelines, Health Level Seven, webpage. Available (accessed on 9.5.2016): https://www.hl7.org/fhir/timelines.html

[33] W3C10 Timeline Graphic, World Wide Web Consortium, image. Available (accessed on: 27.11.2015): https://www.w3.org/2005/01/timelines/timeline-2500x998.png

[34] The Original HTTP as defined in 1991, World Wide Web Consortium, webpage. Available (accessed on 19.12.2015): https://www.w3.org/Protocols/HTTP/AsImplemented.html

[35] Hypertext Transfer Protocol – HTTP/1.0, World Wide Web Consortium, webpage. Available (accessed on 12.1.2016): http://www.w3.org/Protocols/HTTP/1.0/spec.html

[36] Hypertext Transfer Protocol (HTTP) Method Registry, Internet Assigned Numbers Authority, webpage. Available (accessed on 12.1.2016): http://www.iana.org/assignments/http-methods/http-methods.xhtml

[37] I. Grigorik, Chapter 9. Brief History of HTTP in High Performance Browser Networking, O'Reilly Media, 2013. Available (accessed on 18.12.2015): http://chimera.labs.oreilly.com/books/1230000000545/ch09.html

[38] M. Belshe, R. Peon, M. Thomson, Hypertext Transfer Protocol Version 2 (HTTP/2), Request for Comments: 7540, IETF, 2015. Available (accessed on 19.5.2016): https://tools.ietf.org/html/rfc7540

[39] R. Fielding, J. Reschke, Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, Request for Comments: 7230, IETF, 2014. Available (accessed on 12.1.2016): https://tools.ietf.org/html/rfc7230

[40] J. Reschke, The 'Basic' HTTP Authentication Scheme, Request for Comments: 7617, IETF, 2015. Available (accessed on 13.1.2016): https://tools.ietf.org/html/rfc7617

[41] R. Shekh-Yusef, D. Ahrens, S. Bremer, HTTP Digest Access Authentication, Request for Comments: 7616, IETF, 2015. Available (accessed on 25.1.2016): https://tools.ietf.org/html/rfc7616

[42] M. Jones, D. Hardt, The OAuth 2.0 Authorization Framework: Bearer token Usage, Request for Comments: 6750, IETF, 2012. Available (accessed on 10.1.2016): https://tools.ietf.org/html/rfc6750

[43] D. Raggett, Chapter 2 – a history of HTML in Raggett on HTML 4, Addison-Wesley Longman Publishing Co. Inc., Boston, Massachusetts, USA, 1998. Available (accessed on 29.11.2015): http://www.w3.org/People/Raggett/book4/ch02.html

[44] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O'Connor, S. Pfeiffer, HTML5 specification, World Wide Web Consortium, 2014- Available (accessed on 27.11.2015): https://www.w3.org/TR/html5/

[45] Tips for Webmasters - Don't forget to add a doctype, World Wide Web Consortium Quality Assurance, webpage. Available (accessed on 10.2.2016): https://www.w3.org/QA/Tips/Doctype

[46] Document Object Model (DOM), World Wide Web Consortium, webpage. Available (accessed on 29.1.2016): https://www.w3.org/DOM/

[47] History, HTML Living Standard, Web Hypertext Application Technology Working Group, webpage. Available (accessed on 19.5.2016): https://html.spec.whatwg.org/multipage/introduction.html#history-2

[48] R. Cabanier, J. Mann, J. Munro, T. Wiltzius, I. Hickson, HTML Canvas 2D Context, World Wide Web Consortium, webpage. Available (accessed on 20.12.2015): http://www.w3.org/TR/2dcontext/

[49] SVG vs. canvas: how to choose, Microsoft Developer Network, webpage. Available (accessed on 13.1.2016): https://msdn.microsoft.com/en-us/library/gg193983(v=vs.85).aspx

[50] I. Fette, A. Melnikov, The WebSocket Protocol, Request for Comments: 6455, IETF, 2011. Available (accessed on 27.1.2016): https://tools.ietf.org/html/rfc6455

[51] I. Hickson, Server-Sent Events, World Wide Web Consortium, webpage. Available (accessed on 27.1.2016): https://www.w3.org/TR/eventsource/

[52] Server-sent events, HTML Living Standard, World Wide Web Consortium, webpage. Available (accessed on 8.2.2016): https://html.spec.whatwg.org/multipage/comms.html#server-sent-events

[53] H. W. Lie, B. Bos, The CSS saga in Cascading Style Sheets: Designing for the Web, Addison Wesley Longman Publishing Co. Inc., Boston, Massachusetts, USA, 1999. Available (accessed on 5.1.2016): http://www.w3.org/Style/LieBos2e/history/

[54] CSS current work & how to participate, World Wide Web Consortium, webpage. Available (accessed on 6.1.2016): http://www.w3.org/Style/CSS/current-work

[55] At-rule, Mozilla Developer Network, webpage. Available (accessed on 5.1.2016) https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule

[56] Selectors, Mozilla Developer Network, webpage. Available (accessed on 6.1.2016): https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Getting_started/Selectors

[57] A. Rauschmayer, How JavaScript Was Created in Speaking JavaScript, O'Reilly Media, 2014. Available (accessed on 2.5.2016): http://speakingjs.com/es5/ch04.html

[58] A. Rauschmayer, Standardization: ECMAScript in Speaking JavaScript, O'Reilly Media, 2014. Available (accessed on 2.5.2016): http://speakingjs.com/es5/ch05.html

[59] ECMA-262: ECMAScript 2015 Language Specification, Ecma International, 2015. Available (accessed on 2.5.2016): http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf

[60] WeeChat, website. Available (accessed on 23.5.2016): https://weechat.org/

[61] Node.js, website. Available (accessed on 23.5.2016): https://nodejs.org/

[62] JSON, webpage. Available (accessed on 10.1.2016): http://www.json.org/

[63] JSON Schema, website. Available (accessed on 10.1.2016): http://json-schema.org/

[64] ECMA-404: The Json Data Interchange Format, Ecma International, 2013. Available (accessed on 15.12.2015): http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[65] T. Bray, The JavaScript Object Notation (JSON) Data Interchange Format, Request for Comments: 7159, IETF, 2014. Available (accessed on 10.1.2016): https://tools.ietf.org/html/rfc7159

[66] E. Marcotte, Responsive Web Design, A List Apart, No. 306, 2010. Available (accessed on 10.5.2016): http://alistapart.com/article/responsive-web-design

[67] Bootstrap, website. Available (accessed on 12.1.2016): http://getbootstrap.com/

[68] J. D. Meier, D. Hill, A. Homer, J. Taylor, P. Bansode, L. Wall, R. Boucher Jr., A. Bogawat, Architectural Patterns and Styles in Microsoft Application Architecture Guide, Microsoft Developer Network, 2009, webpage. Available (accessed on 22.5.2016): https://msdn.microsoft.com/en-us/library/ee658117.aspx

[69] R. T. Fielding Architectural Styles and the Design of Network-based Software Architectures, University of California, Irvine, 2000. Available (accessed on 7.1.2016): https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[70] Complete basic operations using SharePoint 2013 REST endpoints, Microsoft Developer Network, webpage. Available (accessed on 19.5.2016): https://msdn.microsoft.com/en-us/library/office/jj164022.aspx

[71] R. Fielding, J. Reschke, Hypertest Transfer Protocol (HTTP/1.1): Semantics and Content, Request for Comments: 7231, IETF, 2014. Available (accessed on 8.12.2015): https://tools.ietf.org/html/rfc7231

[72] D. Trowbridge, D. Mancini, D. Quick, G. Hohpe, J. Newkirk, D. Lavigne, Web Presentation Patterns: Model-View-Controller in Enterprise Solution Patterns Using Microsoft .NET, Microsoft Developer Network, 2003. Available (accessed on 20.5.2016): https://msdn.microsoft.com/en-us/library/ms978748.aspx

[73] Patterns for Web Client: Model-View-Presenter in Web Client Software Factory, Microsoft Developer Network, 2010. Available (accessed on 20.5.2016): https://msdn.microsoft.com/en-us/library/ff709839.aspx

[74] Developing a Windows Phone application using the MVVM Pattern: The MVVM Pattern in Phone Development patterns & practices, Microsoft Developer Network, 2012. Available (accessed on 20.5.2016): https://msdn.microsoft.com/en-us/library/hh848246.aspx

[75] D. Shapiro, Web UI Component Architectures in Web Component Architecture & Development with AngularJS, Leanpub, webpage. Available (accessed on 21.5.2016): https://leanpub.com/web-component-development-with-angularjs/read

[76] Developer Guide: Scopes, AngularJS, webpage. Available (accessed on 21.5.2016): https://docs.angularjs.org/guide/scope

[77] API Reference: $rootScope.Scope, AngularJS, webpage. Available (accessed on 21.5.2016): https://docs.angularjs.org/api/ng/type/$rootScope.Scope#$broadcast

[78] Developer Guide: Services, AngularJS, webpage. Available (accessed on 21.5.2016): https://docs.angularjs.org/guide/services

[79] Developer Guide: Directives, AngularJS, webpage. Available (accessed on 21.5.2016): https://docs.angularjs.org/guide/directive

[80] Developer Guide: Components, AngularJS, webpage. Available (accessed on 21.5.2016): https://docs.angularjs.org/guide/component

[81] J. Eldridge, D. Richley, C. Eggett, Clinical Guidelines by Consensus - Recording a standard 12-lead electrocardiogram: An approved methodology by the Society for Cardiological Science and Technology (SCST), The Society for Cardiological Science & Technology, 2010. Available (accessed on 23.5.2016): http://www.scst.org.uk/resources/CAC_SCST_Recording_a_12-lead_ECG_final_version_2014_CS2v2.0.pdf

[82] PhysioBank Annotations, PhysioNet, webpage. Available (accessed on 20.5.2016): https://www.physionet.org/physiobank/annotations.shtml

[83] ANSI/AAMI EC13:2002 – Cardiac monitors, heart rate meters, and alarms, Association for the Advancement of Medical Instrumentation, 2002. Available (accessed on 15.5.2016): http://www.pauljbennett.com/pbennett/work/ec13/ec13.pdf

[84] Animation Frames, HTML Living Standard, The Web Hypertext Application Technology Working Group, webpage. Available (accessed on 2.5.2016): https://html.spec.whatwg.org/multipage/webappapis.html#animation-frames

[85] Monitorix, website. Available (accessed on 23.5.2016): http://www.monitorix.org/

[86] React, website. Available (accessed on 23.5.2016): https://facebook.github.io/react/

[87] Cycle.js, website. Available (accessed on 23.5.2016): http://cycle.js.org/

[88] Ember.js, website. Available (accessed on 23.5.2016): http://emberjs.com/

[89] D3.js – Data-Driven Documents, website. Available (accessed on 23.5.2016): https://d3js.org/

[90] NVD3: Re-usable charts for d3.js, website. Available (accessed on 23.5.2016): http://nvd3.org/

[91] HighCharts, website. Available (accessed on 23.5.2016): http://www.highcharts.com/